UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

PLANIFICADOR MENÚS SEMANALES EN ANDROID

Miguel Olivera Cardo
Tutora: Rosa María Carro Salas

Julio 2020

PLANIFICADOR DE MENÚS SEMANALES EN ANDROID

AUTOR: Miguel Olivera Cardo
TUTORA: Rosa María Carro Salas

GHIA
Dpto. Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Julio de 2020

Resumen (castellano)

Este Trabajo Fin de Grado consiste en el análisis, diseño, implementación y publicación de una aplicación para dispositivos móviles que ejecutan el sistema operativo Android, que consiste en ofrecer al usuario menús de comidas semanales personalizados para cumplir con unos requisitos nutricionales que el usuario pueda necesitar en función de algunas de sus características.

Además de ello, se trata de explicar al usuario por qué se le ofrece el menú sugerido en concreto, para que así mientras utiliza la aplicación pueda obtener unos conocimientos en cuanto a nutrición y alimentación que también le permitan generar sus propios menús semanales.

Asimismo, la aplicación también incorpora elementos de apoyo para gestionar el menú semanal, como una lista de la compra que añade los elementos de forma automática o un buscador de recetas y sigue las recomendaciones y guías de diseño proporcionadas por Google para sus aplicaciones gracias al uso de Material Design.

Dada la naturaleza interactiva de la aplicación, ésta ha sido ideada, diseñada y validada involucrando al usuario final en todas las fases del proyecto, siguiendo un enfoque de desarrollo de software centrado en el usuario.

Actualmente está disponible una versión de la aplicación con la funcionalidad básica implementada y que se puede instalar en dispositivos móviles con Android, a través de Android Studio.

Abstract (English)

This Bachelor Thesis treats the analysis, design, implementation, and publication of an Android application for smartphones powered by the operative system known as Android. This application offers weekly personalized menu recommendations to achieve some nutritional requirements based on some of the characteristics of the user.

In addition, the application tries to explain the user why those menus are suggested, so that, in the future, he can make his own menus based on the knowledge acquired during the usage of the application.

Besides the mentioned functionality, the user can rely on a shopping list that adds the elements automatically in order to cook the recipes, provides a search engine for recipes and follows the design schemes recommended by Google known as Material Design.

As the application has an important interactivity component, final users have participated in the conception, design and validation of the project, following a user-centered design.

At this moment, I have developed an application that covers the basic functionality described above and can be installed through Android Studio, in smartphones running Android.

Palabras clave (castellano)

Android, Android Studio, Cloud Firestore, Diseño, Firebase, Implementación, Menús, MVVM, Nutrición, Realtime Database, Recetas, Recomendación.

Keywords (inglés)

Android, Android Studio, Cloud Firestore, Design, Firebase, Implementation, Menus, MVVM, Nutrition, Realtime Database, Recipes, Recommendation

Agradecimientos

INDICE DE CONTENIDOS

INDICE DE FIGURAS

INDICE DE TABLAS

1 Introducción

En el siguiente capítulo se muestra el motivo por el cual se ha elegido la temática de este TFG y se muestra de forma resumida la organización del documento.

1.1 Motivación

Este TFG busca ofrecer una solución al problema que se presenta a la hora de elegir una alimentación a seguir y organizar las comidas de la semana. Este es un problema al que se enfrenta en su día a día la gran mayoría de la población y a pesar de que en los últimos años el aprendizaje nutricional ha sido más accesible gracias a la expansión del uso de Internet. En este proyecto, además, se intenta explicar por qué se ha escogido cada receta, para que el usuario a la vez pueda entender las recomendaciones que recibe e ir aprendiendo de nutrición, ya que es importante para la salud. (Taejin Jung, 2019)

Esto, unido a cómo los teléfonos inteligentes o smartphones se han convertido en una extensión más de las personas, hace que este sea el medio que más repercusión puede tener para ayudar con problemas en el ámbito de la alimentación, al tener la capacidad de llegar a más usuarios y por su facilidad de uso.

Ambos motivos han impulsado la creación de una aplicación para recomendar menús semanales en función de unas características del usuario, tratando de dar un enfoque educativo, utilizando los teléfonos móviles como medio, en concreto aquellos que utilizan Android como sistema operativo.

1.2 Objetivos

El objetivo principal del trabajo realizado es hacer recomendaciones personalizadas de menús de comidas semanales en función de las características de los usuarios, tratando de añadirle una pincelada educativa, que permita los usuarios entender por qué se realizan determinadas recomendaciones si así lo desean, utilizando para ello dispositivos móviles.

Los objetivos de forma desglosada son los siguientes:

1. Desarrollar un proyecto software centrado en el usuario siguiendo los siguientes pasos:
   - Análisis de requisitos funcionales y no funcionales y posterior validación de requisitos con el usuario.
   - Diseño, maquetación y mapa de navegación junto con la posterior validación del diseño con el usuario.
   - Desarrollo de la arquitectura lógica
   - Codificación
   - Pruebas

2. Dar soporte a una selección automática de comidas a proponer al usuario semanalmente.
3. Ofrecer un enfoque educativo para que el usuario entienda el motivo por el que se realizan dichas recomendaciones.

4. Implementar la aplicación para ejecutarse en dispositivos móviles que utilizan Android como sistema operativo.

1.3 Organización de la memoria

La memoria consta de los siguientes capítulos:

- **Estado del arte:** este capítulo ofrece una visión general de la situación actual en el ámbito de la recomendación de comidas en aplicaciones móviles, observando los puntos fuertes y débiles de cada uno de los ejemplos que se muestran.
- **Análisis:** este capítulo resume los subsistemas identificados junto con sus requisitos funcionales y no funcionales, además de la validación de éstos con los usuarios.
- **Diseño:** este capítulo describe la maquetación de la interfaz gráfica con la validación del usuario, el algoritmo de recomendación y el diseño de la base de datos.
- **Desarrollo:** este capítulo explica cómo se ha llevado a cabo el desarrollo de la aplicación, con sus componentes arquitectónicos y el diseño de la interfaz.
- **Integración, pruebas y resultados:** este capítulo contiene los tests realizados para comprobar el funcionamiento de la aplicación en algunos casos de uso.
- **Conclusiones y trabajo futuro:** este capítulo se incluyen las conclusiones del trabajo realizado y posibles implementaciones a hacer para el futuro.

2 Estado del arte

2.1 Contexto actual

Las opciones de planificación de menús disponibles actualmente para la plataforma Android buscan, en general, ofrecer información de carácter nutricional en forma de platos, en lugar de centrarse en alimentos específicos, por lo que se enfoca hacia una estructuración en forma de dietas que se contemplan como una unidad semanal o quincenal en la que cada día se ofrecen unas recetas al usuario. A pesar de ello, algunas de las aplicaciones disponibles actualmente también tratan de mostrar información nutricional de los alimentos por separado, organizados en base a los principales macronutrientes, que son proteínas, hidratos de carbono y grasas. Este tipo de soluciones son también muy útiles, ya que pueden actuar como un elemento complementario a la planificación o recomendación de menús. Un ejemplo de este tipo de aplicaciones es "Contador de calorías My Fitness Pal".

Otras aplicaciones tratan de complementar la alimentación con un estilo de vida activo, por lo que realizan sugerencias para la realización de ejercicio diario, como la app "8fit – Fitness y Nutrición". Además de ello, también ofrecen rutinas de ejercicios para entrenamiento de fuerza junto con entrenamiento cardiovascular y, en función de dichos ejercicios, se trata de compensar el gasto calórico extra y la ingesta de nutrientes importantes para generar masa muscular como son las proteínas.

Aun así, las aplicaciones en general presentan una limitación en cuanto a su carácter educativo, ya que estás ofrecen información que el usuario puede seguir, pero sin explicar por qué se ha tomado una decisión o se ha realizado una recomendación en cuanto a la dieta, haciendo al usuario dependiente de la aplicación. "Nutrición para entrenamiento" es una aplicación que puede desmarcarse del resto en este aspecto, pero su extrema sencillez y escasez de funcionalidad no la hacen una buena candidata para desarrollar menús.

2.2 Aplicaciones similares

Actualmente la situación respecto aplicaciones de contenido nutricional y generador de menús en Google Play Store es la siguiente:

Tabla 1 App Contador de Calorías My Fitness Pal

Nombre	Contador de Calorías My Fitness Pal
Referencia	https://play.google.com/store/apps/details?id=com.myfitnesspal.android
Ventajas	Desventajas
• Cuantifica las calorías que los usuarios introducen manualmente, así como la cantidad de agua tomada. • Permite establecer objetivos de nutrición y de pasos.	• Obliga a iniciar sesión desde el principio, haciendo que el usuario tenga que dar sus datos si quieres ver que ofrece la aplicación realmente.

Básicamente es como una base de datos que el usuario va creando a lo largo del tiempo, en la que el añade las comidas y demás información. Está enfocada a un cliente que ya tiene cierto conocimiento sobre este tema y que solo quiere cuantificar los nutrientes de su alimentación.

Tabla 2 App Nutrición y Fitness Coach: Dietas y Recetas

Nombre	Nutrición y Fitness Coach: Dietas y Recetas
Referencia	https://play.google.com/store/apps/details?id=com.jappli.nutritionfitness
Ventajas	Desventajas
• Ofrece una sección de favoritos de los planes nutricionales que ofrece, así como de las recetas. • En los planes nutricionales permite marcar el día con las recetas que toquen, para no tener que seguir un orden y poder elegir cualquiera de los 7 menús planteados según lo que le pueda apetecer al usuario del menú en un día concreto. • La sección de recetas incluye un apartado de notas, lo cual facilita que el usuario apunte cosas que quiera recordar una vez cocinada alguno de los platos sugeridos.	• Tiene demasiadas herramientas que sobrecargan la interfaz y que realmente no aportan valor. • En los menús que genera la aplicación para sugerir los platos apenas se ofrecen explicaciones y tampoco se incluyen las recetas para realizarlos, hay que buscarlas luego en la sección de recetas a mano, que además no tiene buscador. • En general no explica por qué un plan es mejor que otro y cuál sería el adecuado para un determinado usuario. • No da opción a tener un perfil del usuario. Es una aplicación genérica que no se adapta a cada usuario.

Opiniones de los usuarios:

- Los usuarios se quejan de que cada plan solo contiene una semana, haciendo que las dietas sean muy repetitivas a lo largo del tiempo.
- En las recetas no se especifica para cuántas personas son los platos.
- No da opción a que el usuario pueda crear sus propios planes en el caso de que ya tuviera los conocimientos de nutrición necesarios.

Tabla 3 App Nutrición para entrenamiento

Nombre	Nutrición para entrenamiento	
Referencia	https://play.google.com/store/apps/details?id=com.insplisity.ultimateworkoutnutrition	
Ventajas		Desventajas
• La aplicación trata de ser más didacta, explicando qué nutrientes contienen algunos alimentos y por qué es más adecuado cocinar algunos platos u otros. • La interfaz es sencilla y muestra la información sin sobrecargarse en exceso.		• Toda la información se ofrece de forma muy teórica, no aporta ejemplos prácticos que el usuario final pueda aplicar. Básicamente es un aglutinador de información.

Opiniones de los usuarios:

- Los usuarios se quejan de que la aplicación no tiene ninguna funcionalidad, ya que básicamente es todo texto y no permite la interacción, por lo que es como un aglutinador de información similar a lo que sería realizar una búsqueda sobre nutrición en Internet.

Tabla 4 App 8fit - Fitness y Nutrición

Nombre	8fit - Fitness y Nutrición	
Referencia	https://play.google.com/store/apps/details?id=com.eightfit.app	
Ventajas		Desventajas
• Nada más abrir la aplicación pregunta por el objetivo que tiene el usuario, su género y su edad. • La interfaz es visualmente agradable y está bien organizada. • Los planes de la versión de pago incluyen vídeos explicativos a la hora de realizar los ejercicios y que aporta cada uno de ellos.		• Ofrece diferentes planes en función de un precio. El plan gratuito solo da acceso a entrenamientos. Para conseguir acceso a un plan personalizado de entrenamiento y de comidas hay que pagar suscripción.

Tabla 5 App FITIA | Nutrición inteligente

Nombre	FITIA | Nutrición inteligente
Referencia	https://play.google.com/store/apps/details?id=com.nutrition.technologies.Fitia&hl=es_419

Ventajas	Desventajas
• Nada más abrir la aplicación pregunta por el objetivo que tiene el usuario, su género y su edad, y recomienda una cantidad de calorías a ingerir en función del peso que el usuario especifica que quiere alcanzar. • Controla la ingesta de nutrientes de una forma visualmente atractiva para monitorizarlos en función de la comida del día.	• Obliga a iniciar sesión desde el principio, haciendo que el usuario tenga que dar sus datos si quiere ver qué ofrece la aplicación realmente. • Pregunta al usuario qué alimentos desea consumir en función del tipo que sean (lácteos, carbohidratos…), por lo que los usuarios menos experimentados o con menos conocimientos sobre este ámbito se quedan fuera. • No ofrece recetas; en concreto, solo plantea algunos alimentos que se podrían tomar para comida del día.

También se han encontrado otras propuestas en el ámbito investigador, como (M. Chen, 2019), en el que se generan pseudo-recetas, de las cuales se extraen sus características para luego buscar recetas reales de una base de datos en función de la similitud. Además, se propone como trabajo futuro integrar recomendaciones con puntuaciones hechas por los usuarios.

Existen también otros trabajos, como (David Elsweiler C. T., 2017), que estudian la viabilidad de sustituir recetas online por algunas más saludables para empujar o sugerir al usuario hacia una mejor alimentación sin dejar de lado sus gustos.

3 Análisis

En este capítulo se especifica la separación de la aplicación en subsistemas y se resumen los requisitos funcionales y no funcionales identificados para cumplir con los requerimientos de la aplicación, organizados por bloques correspondientes a dichos subsistemas.

3.1 Subsistemas

A continuación, se especifican los subsistemas de la aplicación junto con una pequeña descripción de los mismos.

3.1.1 Subsistema de Usuarios

Es el sistema que gestiona los usuarios desde la aplicación. Se encarga de realizar todas las lecturas y escrituras de información sobre los usuarios, su condición física y las puntuaciones de sus recetas en la base de datos, además de realizar las validaciones correspondientes antes de enviar la información de cada usuario al servidor.

3.1.2 Subsistema de Menú

La presentación de los datos obtenidos durante la fase de recomendación es gestionada por este subsistema, que se encarga de recuperar los datos para presentarlos al usuario de forma clara y ordenada.

3.1.3 Subsistema de Búsqueda

Este subsistema dará soporte a la búsqueda de recetas, permitiendo hacer búsquedas por tipos de comida, por ingredientes o directamente platos particulares, permitiendo una consulta detallada de los resultados de forma ordenada.

3.1.4 Subsistema de Recomendación

Este sistema es el que se encarga de escoger las recetas para cada comida de la semana, utilizando un algoritmo de recomendación que trabaja sobre los datos que proporciona el usuario en cuanto a salud y gustos alimenticios.

3.2 Requisitos funcionales y no funcionales

Requisitos funcionales

1. **<SU> Subsistema de Usuarios**
   - 1.1. **<SU> Registro:** El usuario deberá registrarse o iniciar sesión utilizando un email y contraseña o iniciando sesión con su cuenta de Google.
   - 1.2. **<SU> Validación de datos:** El registro comprobará la validez del email, la contraseña y la posible existencia de un usuario con los mismos datos.
   - 1.3. **<SU> Información del usuario:** El usuario proporcionará datos de su condición física para favorecer la personalización de las recomendaciones.
   - 1.4. **<SU> Cancelación de cuenta:** El usuario podrá eliminar sus datos o darse de baja en cualquier momento.
   - 1.5. **<SU> Modificación de datos:** El usuario podrá modificar sus datos con respecto a su condición física.

2. **<SM> Subsistema de Menú**
   2.1. **<SM> Menú semanal:** La aplicación mostrará menús de comidas semanales en función de algunas de las características del usuario.
   2.2. **<SM> Contenido educacional:** Se ofrecerá contenido educativo para que el usuario entienda por qué se recomiendan unos platos u otros.
   2.3. **<SM> Lista de la compra:** Se mostrará una lista de la compra en función de los ingredientes necesarios para cocinar las recetas recomendadas.
   2.4. **<SM> Recetas favoritas:** Se permitirá almacenar recetas favoritas, ya sea desde las búsquedas o desde la recomendación.
   2.5. **<SM> Renovación del menú:** El sistema renovará el menú pasado un intervalo de 7 días desde la última actualización.

3. **<SB> Subsistema de Búsqueda**
   3.1. **<SB> Búsqueda:** El usuario podrá realizar búsquedas de recetas utilizando palabras clave.
   3.2. **<SB> Consulta de resultados:** Se ofrecerá una primera visualización simple de los resultados, permitiendo también acceder a los detalles de cada receta.

4. **<SR> Subsistema de Recomendación**
   4.1. **<SR> Forma física:** El sistema realizará recomendaciones en función de algunos factores físicos del usuario, como su edad, altura, peso, género y nivel de actividad física.
   4.2. **<SR> Gustos alimenticios:** El sistema reconocerá qué tipos de comida son más acordes al usuario y filtrará las recomendaciones en base a ello.

Requisitos no funcionales

- **<RN> Almacenamiento local:** La aplicación utilizará una base de datos relacional para almacenar los contenidos de forma local.
- **<RN> Almacenamiento remoto:** La aplicación almacenará datos en la nube para guardar los datos del usuario de forma permanente.
- **<RN> Obtención de recetas:** La aplicación se apoyará en bases de datos externas de contenido nutricional como por ejemplo SpoonacularAPI.
- **<RN> Interfaz:** El diseño de la interfaz seguirá las directivas de Google para mantener la uniformidad en el diseño y facilitar la usabilidad.
- **<RN> Privacidad:** La seguridad y datos del usuario estará protegido por un sistema de autenticación.
- **<RN> Compatibilidad:** La aplicación deberá tener la posibilidad de ser ejecutado en dispositivos con hardware y software antiguo.
- **<RN> Navegación:** Debido a la gran variedad de formas de navegación disponibles en Android la interfaz orientará la usabilidad de esta a la navegación por gestos implementados por Google.
- **<RN> Arquitectura:** El diseño general de la aplicación seguirá el patrón MVVC (Model-View-ViewModel).
- **<RN> Sin conexión:** La aplicación permitirá consultar las recomendaciones, lista de la compra y recetas favoritas sin necesidad de estar conectado a Internet.

3.3 Validación con el usuario

Al tratarse este TFG de un proyecto software orientado a usuarios, se involucró a usuarios finales en las distintas fases del proyecto. En particular, en el análisis se contó con tres potenciales usuarios de este tipo de aplicación para educir nuevos requisitos y validar los ya identificados, preguntándoles qué considerarían que debería tener una aplicación de este tipo para que resultara útil, e incorporando sus propuestas al conjunto de requisitos identificados inicialmente.

Usuario 1

- O11- Quizá que se pueda registrar con Facebook además de con Google, porque muchas páginas lo permiten y para la gente que no tenga Gmail le sea más cómodo.
- O12- Opción vegetariana/vegana, que está muy de moda.
- O13- Se podrían meter premios o recompensas de algún tipo por fidelidad de uso.
- O14- Que puedas añadir a tus amigos y así poder ver sus recetas y compartir las tuyas si quieres.
- O15- Posible opción de publicar tus recetas en Twitter, Instagram, Facebook…

Usuario 2

- O21- Al registrarse en la aplicación, indicar alergias, intolerancias… y si el menú es orientado a vegetarianos, veganos…
- O22- Al incluir los productos necesarios para la compra, añadir otras opciones en caso de no encontrar ciertos productos para poder elaborar la receta.
- O23- Considerar platos según el tiempo de preparación. Usuarios con poco tiempo en sus días laborales o casos similares preferirán platos que puedan satisfacer sus necesidades alimentarias y que a su vez no conlleven mucho trabajo/tiempo. Por tanto, añadir una sección de filtros con los que poder buscar platos que cumplan ciertos requisitos como este u otros.
- O24- La aplicación permite añadir menús que acompañen a planes de ejercicios de otras fuentes. Para ayudar a esta función, una de las extensiones de la aplicación podría ser ofrecer planes de ejercicios junto a menús específicos, seleccionando estas opciones.
- O25- Como apoyo para cocinar, la aplicación podría contar con herramientas como un cronómetro y similares, o estar vinculada a otras herramientas o aplicaciones que desempeñen esta ayuda.

Usuario 3

- O31- Opción para que las personas puedan subir recetas y el resto de los usuarios puedan descargarlas y añadirlas a sus menús.
- O32- Que muestre recetas conforme a los alimentos que le indiques que tienes por casa.

Algunas de las consideraciones de los usuarios se anotaron para futuras versiones de la aplicación (O14, O31) con el objetivo acotar este TFG.

4 Diseño

En este capítulo se describen las tareas realizadas y sus resultados en la fase de diseño de la aplicación, que abarca, principalmente, el diseño de la interfaz de usuario y la arquitectura de la aplicación, incluyendo los algoritmos que van a permitir el correcto funcionamiento de la aplicación y el diseño de la base de datos correspondiente.

4.1 Maquetación

A continuación, se muestra una primera maqueta que muestra cómo será la interfaz de usuario y la navegación entre pantallas. Para el diseño de la interfaz gráfica se ha utilizado la herramienta de creación de *wireframes* llamada Balsamiq.

4.1.1 Inicio de sesión

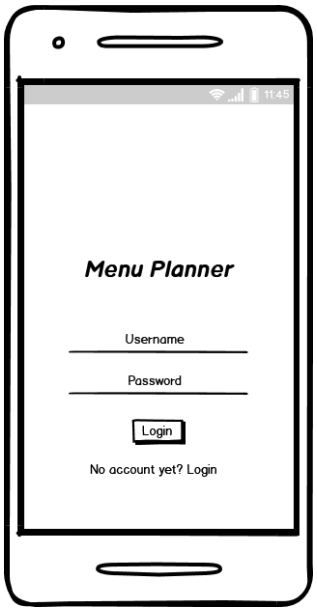


Figura 4-1: Maqueta Inicio de Sesión

Esta pantalla será lo que verán los usuarios la primera vez que abran la aplicación. En ella podrán introducir sus credenciales para iniciar sesión o crear una nueva cuenta si no lo han hecho previamente. Este proceso permitirá dar de alta a los usuarios y almacenar sus datos para poder recuperarlos en caso de que en el futuro deseen volver a la aplicación.

4.1.2 Recopilación de información

Cuando el usuario haya iniciado sesión, pasará a esta pantalla, en la cual introducirá información personal necesaria para realizar las recomendaciones, ya que estás se basan en sus características personales.

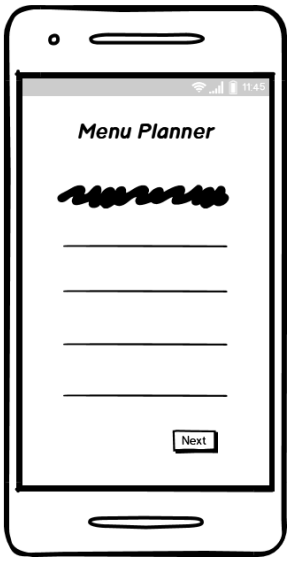


Figura 4-2: Maqueta Recopilación de Información

4.1.3 Menú generado

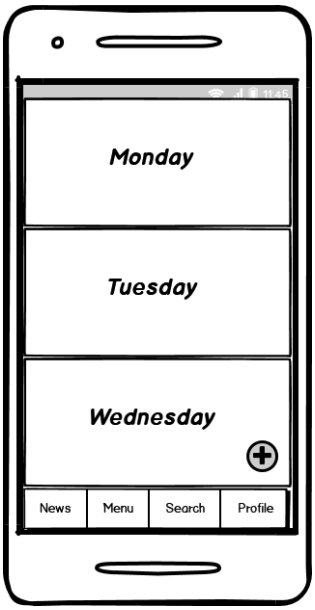


Figura 4-3: Maqueta Menú Semanal

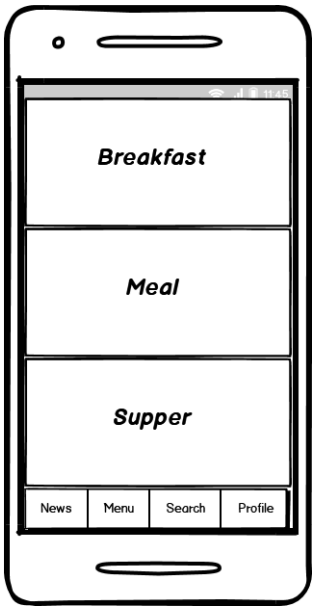
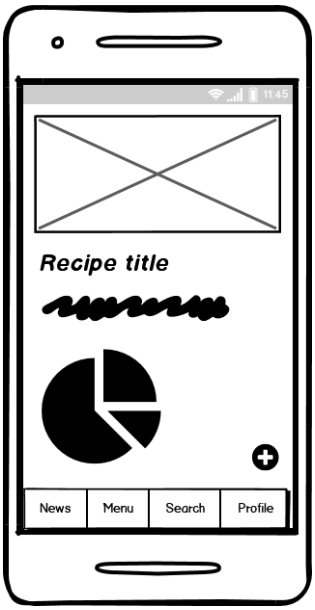


Figura 4-4: Maqueta Menú Diario

Una vez terminado el proceso de registro o si el usuario ya había iniciado sesión, aparecerá la pantalla que muestra los menús de la semana en forma de tarjetas, junto con una barra de navegación inferior. Pulsando en cualquiera de los días, se mostrará otra pantalla con la misma distribución de elementos, pero con distinto contenido, en concreto, las recetas para realizar los platos.

4.1.4 Receta



La receta aparecerá de forma detallada, junto con su información nutricional y una pequeña explicación para que el usuario pueda entender por qué dicha receta es adecuada según sus características particulares. Para acceder a esta información se hará una pulsación en cualquiera de las recetas de la pantalla anterior.

Figura 4-5: Maqueta de Receta Detallada

4.1.5 Lista de la compra

Para acceder a la lista de la compra, el usuario tendrá que acceder a su perfil a través de la barra de navegación inferior y allí podrá acceder a dicha lista. En ella podrá marcar y desmarcar los ingredientes.

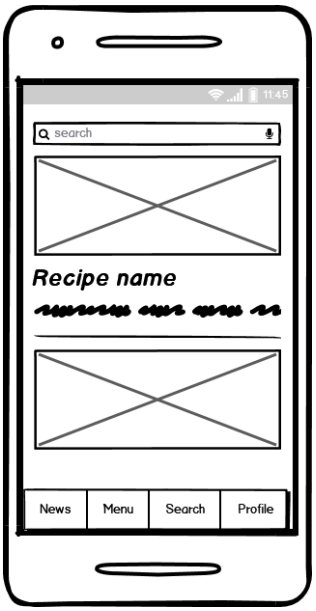
Figura 4-6: Maqueta Lista de la Compra

4.1.6 Panel de control del usuario

En el panel de control del usuario, éste podrá consultar su información y modificarla. También se mostrarán sus recetas favoritas y su lista de la compra. De nuevo, podrá acceder a ella utilizando la barra inferior.

Figura 4-7: Maqueta Panel de Control del Usuario

4.1.7 Motor de búsqueda



En la siguiente pantalla se podrán realizar búsquedas de recetas en la base de datos de la aplicación.

Figura 4-8: Maqueta Resultados de Búsqueda

4.2 Validación con el usuario

El diseño de la interfaz fue validado con los usuarios finales a quienes se involucró para las distintas fases de este proyecto.
Estos usuarios están habituados a usar teléfonos inteligentes en su día a día y cubren el rango de edad desde los 25 hasta los 62 años.
Se simularon las siguientes acciones con los usuarios:

- Iniciar sesión y rellenar el formulario con la información personal.
- Consultar una receta de cualquiera de las recomendadas.
- Modificar algún ingrediente de la lista de la compra.
- Consultar alguna de las recetas guardadas como favoritas.

Durante la validación, en caso de que algún usuario no entendiese la presencia de algún botón o cómo realizar una acción determinada, se le explicaba en el momento. Finalmente, se solicitó a cada usuario que confirmara, para cada tarea realizada durante la simulación, si le había resultado intuitivo o no realizarla, si se había sentido cómodo con la interfaz, si encontró que la disposición de los elementos en las pantallas tenía sentido o era mejorable, y qué le gustaría que fuera de otro modo en la aplicación.
Tras validar el diseño de la interfaz con los usuarios se llegó a las siguientes conclusiones:

- Sería útil añadir una especie de pop-up para previsualizar los resultados de las búsquedas de forma rápida.
- No se entiende muy bien cómo llegar a la lista de la compra, sería bueno añadir un acceso directo desde la receta.

- También convendría añadir acceso directo a la lista de la compra en la barra inferior de la aplicación.
- Podría ser útil separar la lista de la compra en semanas/días/comidas.
- El símbolo + de la sección de menús genera confusión, mejor quitarlo de ahí y dejarlo solo en la receta para permitir sustituir ingredientes, por ejemplo.
- En las recetas, en lugar de añadir un botón + para añadir anotaciones, incluir un campo de texto como un formulario.
- Incorporar botón para editar la receta en vivo y actualizar automáticamente la lista de la compra.
- Añadir sugerencias en la receta de un ingrediente alternativo por si el que aparece en la receta no se tiene por casa.

4.3 Arquitectura Lógica

A continuación, se detallan los elementos que constituyen la arquitectura lógica de la aplicación, que proporcionan datos a la vista y permiten el correcto funcionamiento de la aplicación.

4.3.1 Algoritmo de Recomendación

El algoritmo de recomendación consiste en la realización de unos cálculos que darán lugar a una puntuación final que será utilizada para asociar el perfil de un usuario con un menú determinado.

El algoritmo secciona la tarea de recomendar en tres pasos principales:

1. Establecer requisitos nutricionales del usuario basado en Basal Metabolic Rate (BMR). (David Elsweiler M. H., 2015)
2. Estimar puntuaciones para las recetas, los requisitos y las preferencias del usuario.
3. Comprobar la validez del menú generado.

1. Establecer requisitos nutricionales del usuario

Para determinar los requisitos nutricionales se necesita realizar un cálculo del BMR, para saber cuál es el consumo calórico del usuario. Para realizar este cálculo se aplica la fórmula de Harris-Benedict revisada, que toma el peso en kg, la altura en cm y la edad en años (Harris-Benedict, n.d.).
En función del género, la función a aplicar es la siguiente:
Hombre:

BMR- 88.362 + (13.397 x peso en kg) + (4.799 x altura in cm) – (5.677 x edad en años)

Mujer:

BMR- 447.593 + (9.247 x peso en kg) + (3.098 x altura en cm) – (4.330 x edad en años)

Una vez obtenido el BMR se debe ajustar en función del nivel de actividad física del usuario. Para ello se multiplica el valor calculado previamente por los siguientes valores:

- Sedentario: BMR x 1.2
- Moderadamente Activo: BMR x 1.55
- Activo: 1.725

Para mantener una dieta balanceada a nivel nutricional se establece que un 45% de las calorías consumidas deben proceder de carbohidratos, un 25% de grasas y un 30% de proteínas. Por último, se establecerá un margen de error del 10% en cuanto a las calorías para hacer la selección de recetas más flexible.

2. **Estimar puntuaciones para las recetas, los requisitos y las preferencias del usuario.**

Para estimar las puntuaciones de las recetas se generan grupos de tres comidas, de tal forma que el consumo calórico diario quede repartido. Para hacer un reparto de forma coherente se extrapola la cantidad de calorías basándose en un menú estándar. (Globalnews, n.d.)
De esta forma el desayuno abarca el 0,178% de las calorías totales, la comida el 0,47% y la cena el 0,352%. La tabla 6 muestra la distribución de calorías para cada comida en función de las calorías totales a consumir durante un día.

Tabla 6 Distribución de calorías de cada comida

Calorías	1200	1500	1800	2100	2400	2700	3000
Desayuno	213,6	267	320,4	373,8	427,2	480,6	534
Comida	564	705	846	987	1128	1269	1410
Cena	422,4	528	633,6	739,2	844,8	950,4	1056

Y el reparto de los principales macronutrientes en función de las calorías diarias a consumir quedaría como se muestra en la tabla 7. (Uccs)

Tabla 7 Distribución de macronutrientes en función de calorías

Calorías	30% Proteínas (g)	25% Grasas (g)	45% Carbohidratos (g)
1200	90	33	135
1500	112	42	75
1800	135	50	202
2100	158	58	236
2400	180	67	270
2700	202	75	303
3000	225	83	338

Con esto se consigue aplicar una serie de restricciones a las recetas para conseguir un nivel en principio saludable, con la mayor precisión posible. Pero también es interesante sugerir a los usuarios recetas que además les puedan gustar. Para ello se utiliza un algoritmo de recomendación para elegir las recetas finales, que trabaje junto con lo calculado anteriormente. Para ello se plantearon tres algoritmos: kNN basado en usuario, kNN basado en ítem y recomendación basada en contenido.

A pesar de que todos ellos tienen un arranque en frío desfavorable, y esta va a ser la situación de la aplicación al principio, ya que no se dispondrán de usuarios ni ítems iniciales, la recomendación basada en contenido es la que ofrece menos desventajas. Además de ello, los algoritmos kNN basados en vecindarios resultan más costosos computacionalmente a la hora de calcular las similitudes entre usuarios o entre ítems, y encontrar recetas puntuadas por k usuarios para generar los vecindarios va a ser complicado inicialmente, por ejemplo en el siguiente artículo, (Vivek M.B., 2018) , se utilizan vecindarios de 200 usuarios para hacer una recomendación válida. Además de ello, se debe añadir el coste computacional de

encontrar el tamaño del vecindario más conveniente ya que no es lo más optimo aplicar un valor k fijo a diferentes conjuntos de datos, por lo que se añadiría otra tarea que consumiría tiempo y reduciría la velocidad de respuesta (Shichao Zang, 2017). Al tratarse de una aplicación para teléfonos móviles, la velocidad cobra mayor importancia.
Por último, los algoritmos de recomendación basados en contenido se complementan bien en implementaciones híbridas, por lo que, en el futuro, ejecutando estas tareas en un servidor externo, como puede ser Firebase Functions y al poseer de una fuente de datos de usuarios o ítems de un tamaño suficiente, se podría implementar alguno de los algoritmos kNN y observar si el sistema mejora su precisión utilizando métricas de evaluación.

A continuación, se muestra el algoritmo escogido basado en recomendación por contenido (Teh Lee Cheng, 2014) con algunas adaptaciones para poder usarse en la aplicación desarrollada para este TFG.
Cada receta es puntuada por el usuario, con un peso que oscila entre 1 y 5. Si el usuario guarda una receta como favorita, se asignará una puntuación de 5 automáticamente. Cada receta pertenece a un grupo en función de la hora para la que está enfocada la receta para ser consumida (desayuno, comida, o cena) y de su origen (por ejemplo, comida italiana o española). Por lo tanto, se genera una separación por tipos de comida y en cada uno de ellos habrá un tipo de comida determinado por su procedencia, que será utilizado como una condición más a la hora de buscar las recetas.
Una vez se obtienen las puntuaciones, se calcula el peso de cada categoría, para saber qué característica es más afín al usuario y hacer recomendaciones en base a ello. Para calcular el peso se utiliza la siguiente fórmula, donde *i* es el grupo, C_j es la puntuación de cada receta del grupo, C_i es la puntuación de la receta a estudiar y W_i es el peso calculado de dicha receta, siendo n el número de categorías en el grupo.

$$W_i = \frac{C_i * \sum_{j=0}^{n} C_j}{n}$$

Ya con la cantidad de calorías calculadas, los requisitos de macronutrientes necesarios de acuerdo con las calorías y las características de las recetas que al usuario puedan gustar, se buscan las recetas para generar el menú.

El algoritmo queda resumido en el diagrama mostrado en la figura 4-9.

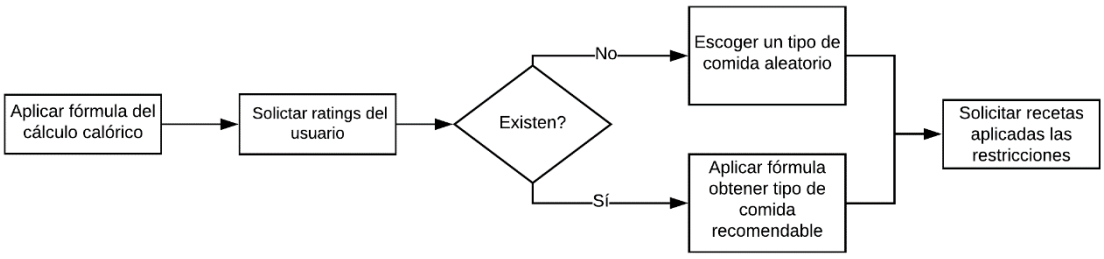


Figura 4-9 Algoritmo de recomendación

3. **Comprobar la validez del menú generado.**

Para asegurar que el menú es válido se comprobará que en más de dos días no se repite el mismo conjunto de tres recetas, sin importar el orden, y se permitirá repetir algunos platos durante la semana pensando en la reutilización de comida cocinada durante la semana.

4.3.2 Bases de datos

La aplicación almacena todos los datos utilizando dos bases de datos, una NoSQL en remoto y otra SQL en local. Para remoto se utiliza una plataforma en la nube desarrollada por Google que puede ser usada para aplicaciones móviles y servicios web. En concreto, para este desarrollo se utilizan dos de sus servicios, Firebase Authentication y Firebase Database. Para el almacenamiento local se utiliza la librería Room que implementa una base de datos en SQLite.

El servicio remoto permite desarrollar un sistema de inicio de sesión y almacenamiento de datos de usuario necesario para su identificación, donde el email e identificador son visibles para el administrador. Para el inicio de sesión se ofrecen numerosas alternativas y distintos servicios, pero para el desarrollo de este TFG se han elegido el inicio de sesión por email y contraseña o acceder con una cuenta de Google, la cual es necesaria para descargar aplicaciones en la Play Store, por lo que todos los usuarios podrían utilizar este método, que les permite iniciar sesión o registrarse con una sola pulsación y sin necesidad de introducir contraseñas. Una vez que un usuario crea una cuenta por cualquiera de los dos métodos mencionados, el sistema genera un número identificativo único para cada usuario, que se puede utilizar para relacionar los datos posteriormente en la base de datos remota.

En cuanto a la base de datos, Firebase ofrece dos alternativas, Firebase Realtime Database y Firebase Cloud Firestore. El primero de ellos fue el primer servicio de almacenamiento lanzado en esta plataforma, el cual estructura sus datos en un JSON. Por otro lado, Firebase Cloud Firestore, es un servicio que utiliza una base de datos NoSQL basado en colecciones y documentos.

Hoy en día, la recomendación general de Google es utilizar Cloud Firestore, su servicio más reciente, que recibirá soporte durante más tiempo y tiene un sistema de consultas más potente, por delante de Realtime Database. Existe un caso de uso para el que recomiendan Realtime Database y es para aquellos servicios que necesiten de hacer actualizaciones asíncronas de forma continua con la base de datos, ya que dicho servicio posee mejores latencias que Cloud Firestore. Como este argumento no es relevante para el desarrollo de esta aplicación y una estructuración en forma de colecciones y documentos es más eficiente que un JSON, se ha decidido utilizar Cloud Firestore.
A continuación, se muestran los esquemas de diseño de ambas bases de datos y el uso específico para cada una de ellas.

4.3.2.1 Base de datos local

La aplicación utiliza la librería Room para gestionar una base de datos local. En concreto, la base de datos tiene tres tablas principales que almacenan la información sobre las

recetas, los ingredientes y el usuario. Esta base de datos tiene el objetivo de actuar como una caché para los datos del usuario y para el menú generado, con el fin de obtener una velocidad óptima en cuanto al acceso de datos.
A continuación, en la figura 4-10, se muestra el esquema de la base de datos local.

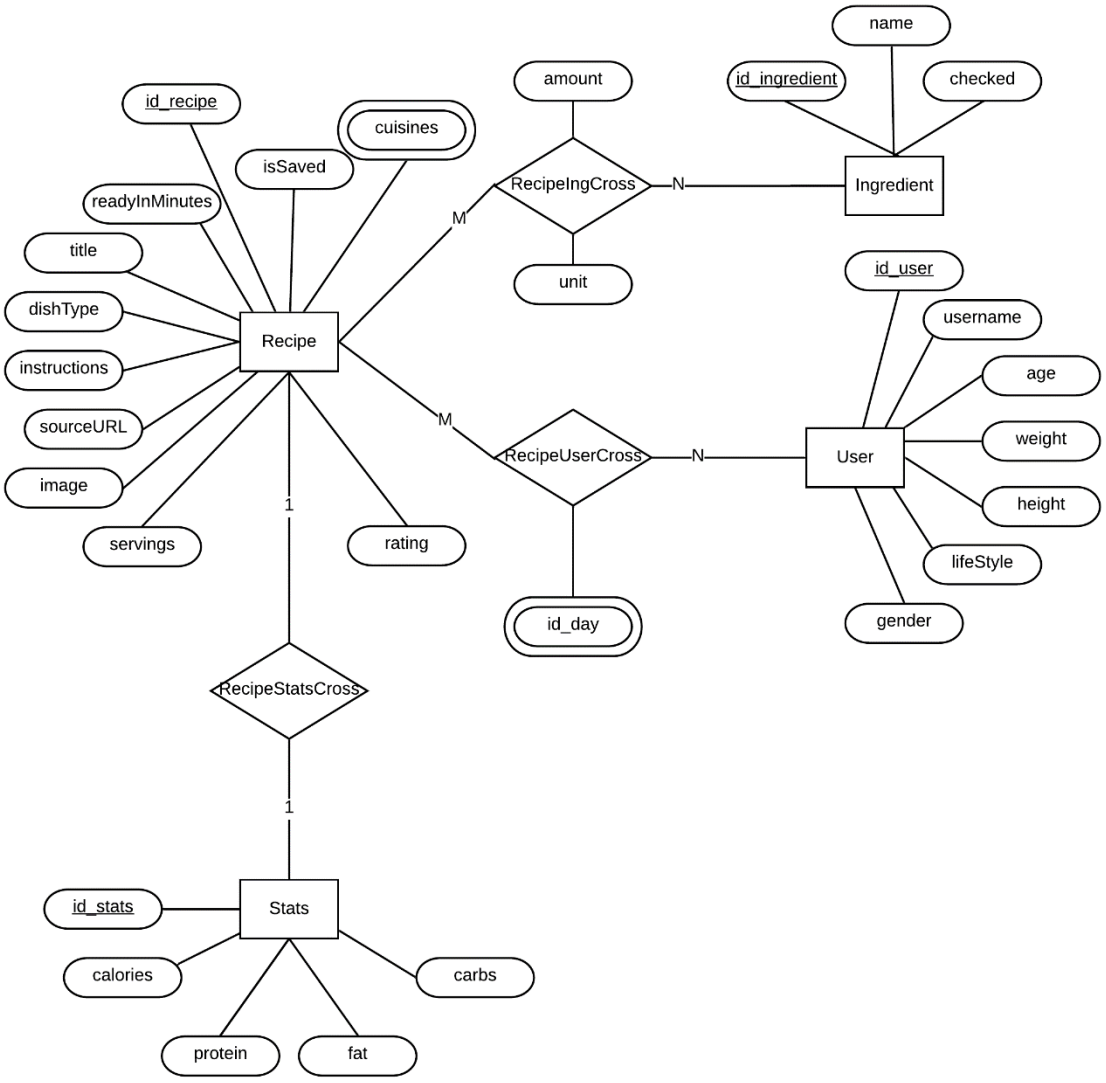


Figura 4-10 Esquema Entidad-Relación de la base de datos local

De esta forma, la tabla *User* será populada con datos cuando el usuario inicie sesión, ya sea con datos nuevos o con datos de Firestore que permitirán implementar funcionalidad de autocompletado para usuarios que ya hayan utilizado la aplicación previamente.
Por otro lado, las tablas de *Recipe* e *Ingredients* obtendrán sus datos de una API llamada Spoonacular que se comentará posteriormente, en el capítulo de Desarrollo. Estos datos serán eliminados después de 7 días, para utilizar el nuevo menú generado después de una semana y cumplir con los términos y condiciones de la API en cuanto a almacenamiento de sus datos.

4.3.2.2 Base de datos remota

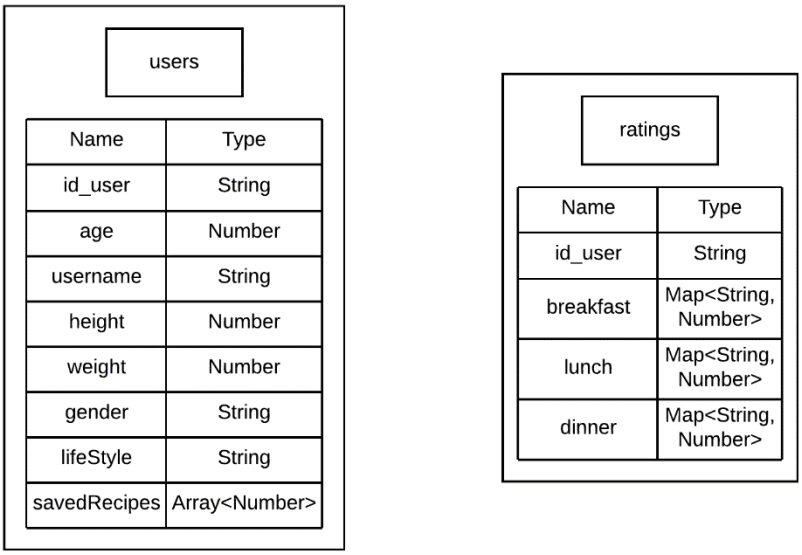


Figura 4-11 Esquema de colecciones de la base de datos remota

La base de datos remota utiliza NoSQL, organizando la información en documentos y colecciones. Esta consta de una serie de colecciones principales:

- ***users***: colección que almacena todo el contenido relativo a los usuarios, como por ejemplo los datos personales que son utilizados para calcular el consumo calórico del usuario.
- ***ratings***: colección que almacena las puntuaciones para cada característica en función del usuario utilizadas para realizar las recomendaciones.

Los documentos de la colección *users* contienen los atributos mostrados en la tabla 6:

Tabla 8 Documento de la colección users

users	Tipo de dato	Descripción
id_user	String	Representa el id del usuario
username	String	Representa el nombre del usuario
weight	Number	Representa el peso del usuario en kg
height	Number	Representa la altura del usuario en cm
age	Number	Representa la edad del usuario
gender	String	Representa el género del usuario
lifeStyle	String	Representa la cantidad de ejercicio físico que realiza el usuario
savedRecipes	Array<Number>	Representa los identificadores de las recetas guardadas como favoritas.

Por último, la colección que almacena todas las características de las recetas para recomendar a los usuarios se llama *ratings*. Está colección se podría colocar como una

entrada más del documento del usuario, pero como los documentos de *ratings* van a estar creciendo en tamaño, he decidido crear una colección para evitar superar el límite de 1MB por documento y darle el mayor límite de almacenamiento posible, a pesar de tener que hacer una lectura y escritura extra por cada nueva característica de un documento de cada usuario. Tampoco tiene sentido colocar este tipo de documentos como una subcolección de *users*, ya que se estaría utilizando una colección solo para un documento nuevo. La estructura de los documentos de la colección se muestra en la tabla 7:

Tabla 9 Documento de la colección ratings

ratings	Tipo de dato	Descripción
id_user	String	Representa el id de la receta
breakfast	String	Contiene un mapa que relaciona características del grupo con una puntuación
lunch	String	Contiene un mapa que relaciona características del grupo con una puntuación
dinner	String	Contiene un mapa que relaciona características del grupo con una puntuación

5 Desarrollo

En el siguiente apartado se incluye información sobre las librerías utilizadas, patrones de diseño, arquitectura del sistema y demás decisiones tomadas para y durante el desarrollo de la aplicación.

Android ofrece a los desarrolladores una colección de librerías para implementar aplicaciones Android, llamada Android Jetpack, la cual ha sido utilizada para desarrollar este TFG, además de algunas otras librerías implementadas por terceros. Al describir los subsistemas de la aplicación se mencionarán muchas de estas librerías y por qué se han utilizado.

5.1 Arquitectura de la aplicación

Como se ha mencionado anteriormente, la arquitectura de la aplicación sigue el modelo MVVM, es decir, Model-View-ViewModel. De esta forma se consigue realizar una separación de los componentes que permiten modularizar el sistema y se definen las comunicaciones entre dichos módulos.
La arquitectura está definida por tres elementos principales:

- **Actividad/Fragmento**: representa a la vista, su función es mostrar al usuario la información que obtiene a través de LiveData.
- **ViewModel**: representa al ViewModel y se encarga de solicitar los datos al repositorio, para posteriormente avisar a la vista cuando se obtiene la información, para así mostrarla en pantalla.
- **Repositorio**: representa al modelo y se encarga de solicitar los datos. En función de la situación en la que se encuentra la aplicación, los solicita de una caché generada con Room, de una base de datos en NoSql proporcionada por Firebase o de una API llamada Spoonacular.

A continuación, se muestra un esquema general de la arquitectura en la Figura 5-1.

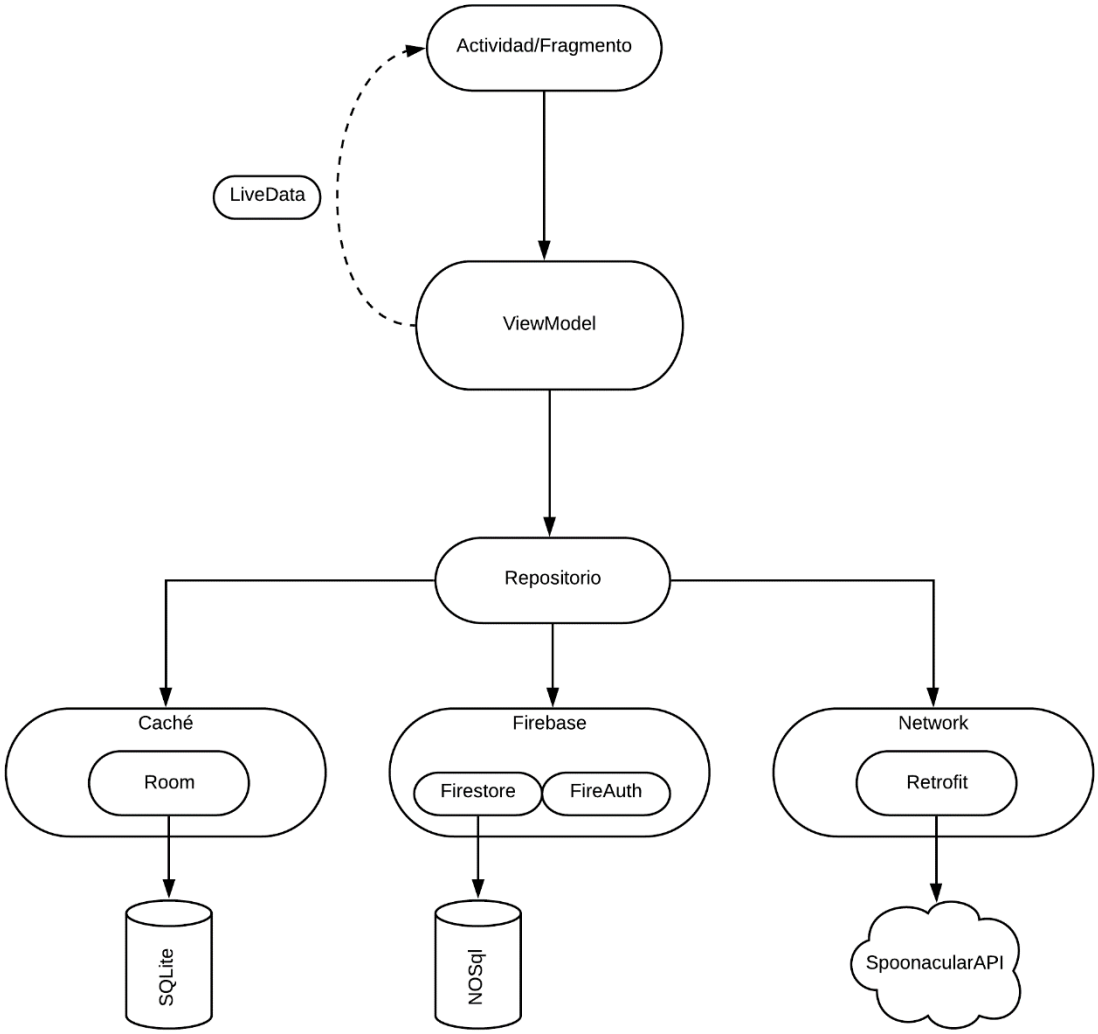


Figura 5-1 Arquitectura del Sistema

A continuación, se explican detalladamente los tres elementos de la arquitectura y como se han realizado.

5.1.1 Actividad/Fragmento

Android ofrece dos formas de implementar la vista para sus aplicaciones; por un lado, existen las actividades y por otro los fragmentos.
Una actividad es el componente que el sistema operativo Android lanza siempre que se inicia una aplicación, pudiendo ésta tener múltiples actividades o solo una. La actividad proporciona la vista dibujando todos los elementos en pantalla.
Por otro lado, están los fragmentos, que también representan una vista, pero con mayor flexibilidad que las actividades. Dentro de una actividad puede haber uno o varios fragmentos.

Ambos tienen en común que trabajan con un ciclo de vida, no exactamente igual, pero muy similar, que les permite crearse o destruirse según el estado en el que se encuentra la aplicación.

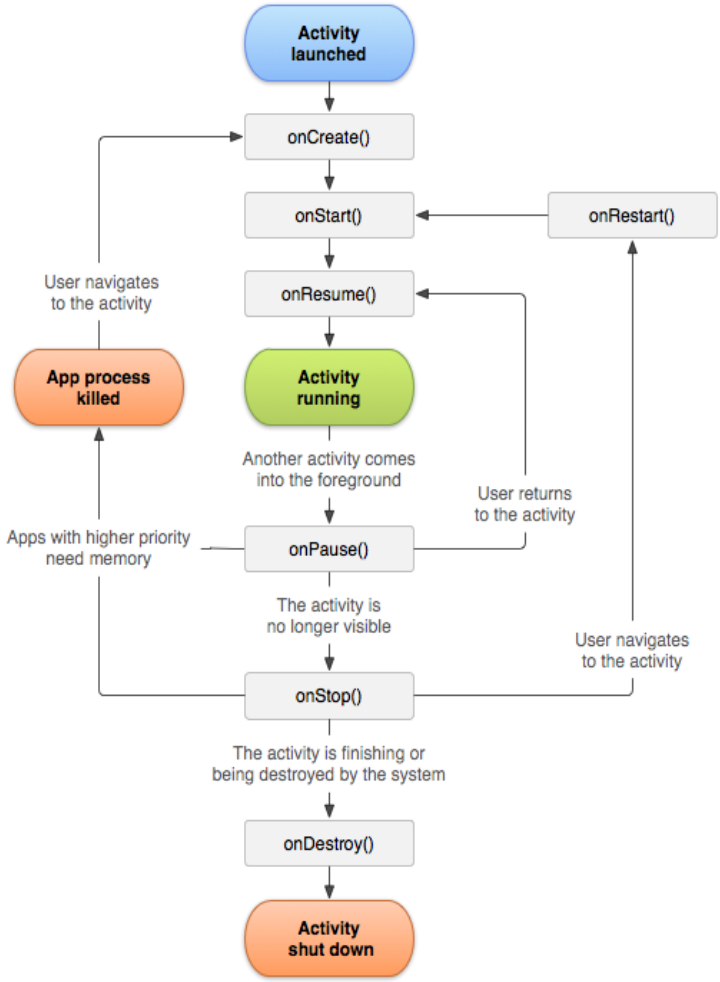


Figura 5-2 Ciclo de vida del componente actividad (Android Developers, s.f.)

Cada estado del ciclo de vida está representado por una función que se dispara cada vez que la actividad entra en un nuevo estado. Es importante tener en cuenta este ciclo de vida, ya que por ejemplo, a la hora de aplicar el patrón Observer, a través de LiveData, se asocian los observadores al ciclo de vida de la vista, de forma que cuando la actividad entra al estado onPause() u onStop(), los observadores liberen sus recursos. Otro ejemplo de la importancia del ciclo de vida es durante un cambio de configuración, como puede ser realizar una rotación del dispositivo o lanzar la aplicación en modo multiventana. En estos casos la actividad pasa por onStop(), onRestart() y onStart(), por lo que si no se guarda el estado de la aplicación, el usuario puede perder el trabajo realizado en la actividad o algún proceso en segundo plano se puede ver interrumpido incorrectamente, provocando el cierre total de la aplicación.

Por otro lado, está el ciclo de vida de un fragmento, que tiene diferentes estados, pero al final trabaja sobre los mismos principios que los del ciclo de vida de una actividad. La figura 5-3 muestra el ciclo de vida de dichos fragmentos.

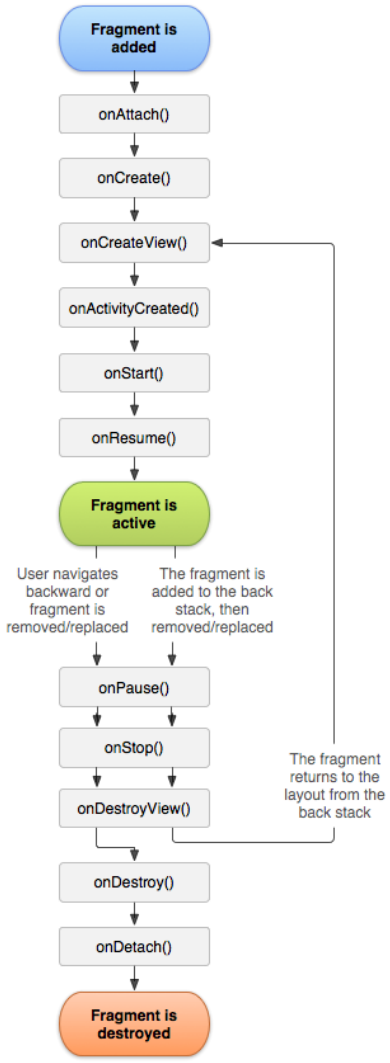


Figura 5-3 Ciclo de vida del componente fragmento (Android Developers, s.f.)

Para el desarrollo de este trabajo se implementa un patrón de diseño llamado “Single Activity”. Este consiste en utilizar una única actividad que se mantiene viva durante todo el ciclo de vida de la aplicación y que actúa como un contenedor para los diferentes fragmentos que van a representar las vistas, que se van intercambiando entre sí a medida que el usuario navega entre las diferentes pantallas de la aplicación, otorgando una alta flexibilidad a la hora de diseñar las vistas de la aplicación y permitiendo que éstas sean reutilizadas fácilmente.
Además, está implementación habilita el uso del componente Navigation, que define las relaciones y transiciones entre fragmentos, haciendo que el desarrollador tenga control sobre la pila de fragmentos.
Uno de los principales argumentos para usar un modelo con varias actividades, es el de tener dos instancias mostrando, por ejemplo, dos recetas diferentes al mismo tiempo, que puedan ser accesibles mediante el gestor de multitarea de Android, pero en este caso no es uno de los requisitos a cubrir, por lo que el patrón de “Single Activity” tiene más sentido.

Además, uno de los puntos fuertes de este diseño es que facilita la adaptación de la interfaz a pantallas de mayor tamaño, como podría ser una tableta o un teléfono plegable.
En un smartphone convencional, la pantalla con los días de la semana y las recetas para cada día supondría tener que realizar una acción, tal y como se ilustra en la parte derecha de la figura 5-4.

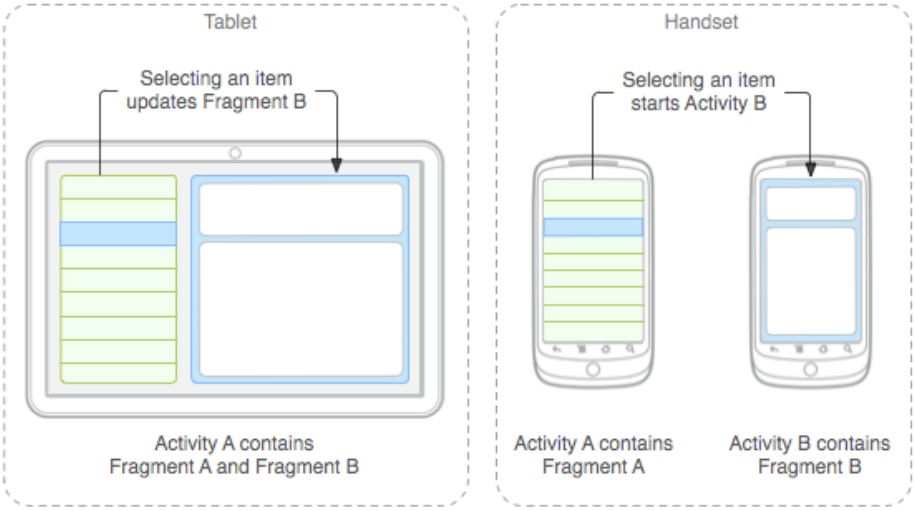


Figura 5-4 Ejemplificación de la flexibilidad de los fragmentos (Android Developers, s.f.)

Por otro lado, si la aplicación se adaptara para trabajar en dispositivos con mayor tamaño de pantalla, se podría aprovechar este espacio, mostrando ambas pantallas de forma simultánea, ahorrando pulsaciones al usuario y evitando así rediseñar la interfaz gracias a la reutilización de fragmentos, tal y como se ilustra en la parte izquierda de la misma figura.

5.1.1 ViewModel

Un ViewModel es un componente que permite almacenar y gestionar datos, siendo consciente del ciclo de vida de la actividad o fragmento al que esté asociado. En él se realizan las tareas asíncronas y se implementan objetos observados por la vista.

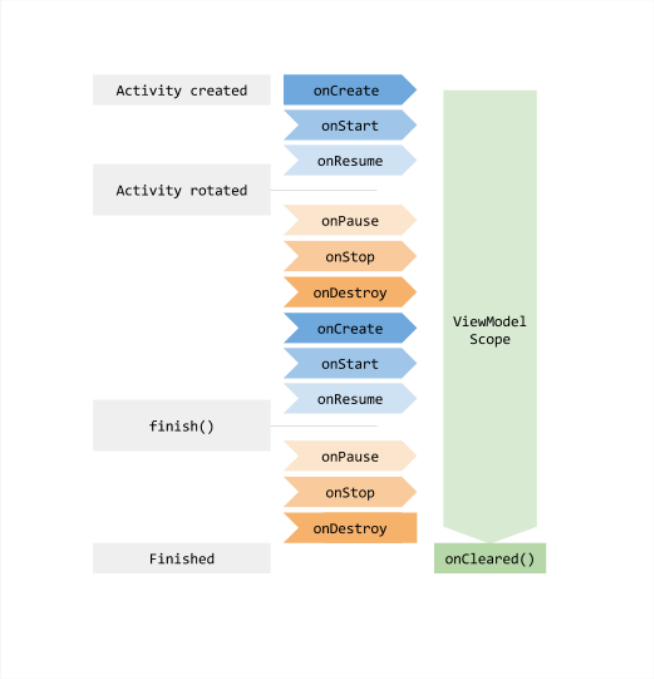


Figura 5-5 Ciclo de vida del componente ViewModel (Android Developers, s.f.)

El ciclo de vida del ViewModel (mostrado en la figura 5-5) permite que, a diferencia de los fragmentos o las actividades, no sea destruido ante un cambio de configuración, sino que se mantiene activo hasta que se alcanza el estado onDestroy() en caso de una actividad o onDettach() en caso de un fragmento. Esta propiedad hace posible que procesos asíncronos o en segundo plano no se vean interrumpidos, viéndose perdido su progreso o que datos ya cargados no desaparezcan y tengan que volver a recuperarse.

También permiten al desarrollador controlar la construcción del objeto ViewModel, a través del patrón Factory, para así añadir inyección de dependencias.

5.1.1 Repositorio

Para la implementación de un repositorio, Android no ofrece un componente concreto, como ocurre con la vista o el ViewModel, sino que ofrece varias librerías para que los desarrolladores realicen sus propias implementaciones que se comuniquen con un repositorio que actúa como nexo entre el ViewModel y los datos.
Este repositorio sirve para implementar las comunicaciones con la base de datos local y remota y con la API de Spoonacular.
Como se ha mencionado anteriormente, la base de datos local está implementada con la librería Room siguiendo el esquema especificado en la Figura 4-11; la base de datos remota sigue el esquema de la Figura 4-12; y la autenticación utiliza Firebase. Por último, las recetas se obtienen de una API, llamada Spoonacular, cuyas solicitudes se han implementado utilizando la librería Retrofit.
Spoonacular proporciona la base de datos para consultar las recetas, además de otras funcionalidades adicionales como la generación de menús, información nutricional, filtros por alérgenos, tipos de dietas, etc. Devuelve sus resultados en formato JSON que se parsea a un objeto con el que la aplicación puede trabajar y convertir al modelo de datos de la base de datos local.

Las acciones de lectura y escritura en el repositorio son tareas costosas y asíncronas, por lo que no se pueden ejecutar en el hilo principal de la aplicación. Este hilo es el encargado de reconocer las pulsaciones del usuario, rellenar las vistas, refrescar las pantallas cada 16 ms, etc. Para solucionar este problema he utilizado las corrutinas de Kotlin, que permiten trabajar con hilos fácilmente y hacen que el código asíncrono sea más sencillo, ya que elimina la necesidad de usar *listeners*.
Estas corrutinas trabajan con hilos precargados que están diseñados para tareas específicas. Para esta aplicación se utilizan los hilos de tipo Main, que son de uso general y los de tipo IO, para tareas de lectura y escritura. A continuación, se muestra un ejemplo de implementación de estas corrutinas para realizar una operación asíncrona.
En el ViewModel se definen los siguientes hilos, de tipo IO.

```
private var viewModelJob = Job()
private val uiScope = CoroutineScope( context: Dispatchers.IO + viewModelJob)
```

Figura 5-6 Definición del contexto de corrutina

También se especifica que cuando el ViewModel se elimine, los hilos de este contexto dejen de ejecutarse, para no consumir recursos.

```
override fun onCleared() {
    super.onCleared()
    viewModelJob.cancel()
}
```

Figura 5-7 Liberación de corrutinas en el ViewModel

En la figura 5-8 se muestra cómo se lanzan:

```
fun saveRating(recipe: Recipe, rating: Int) {
    uiScope.launch { this: CoroutineScope
        repository.setRecipeRating(recipe, rating)
    }
}
```

Figura 5-8 Ejemplo de lanzamiento de una corrutina

Y en el repositorio se define una función de tipo *suspend* para que pueda ser lanzada en el contexto de una corrutina. En este caso se lanza otro hilo, que está enlazado al contexto anterior, para que se realice de forma paralela la escritura en la base de datos local y la escritura en la base de datos remota.

```
override suspend fun setRecipeRating(recipe: Recipe, rating: Int) {
    CoroutineScope(IO).launch { this: CoroutineScope
        appDatabase.appDao.updateRecipeRating(recipe.id_recipe, rating)
    }
    firebase.setRating(recipe.dishTypes, recipe.cuisines, rating)
}
```

Figura 5-9 Definición de una función de tipo suspend

A continuación, en la figura 5-10, se muestra un ejemplo de cómo las corrutinas hacen más fácil la lectura de código asíncrono. La siguiente función solicita un documento con los datos de un usuario en la base de datos remota:

```
suspend fun getUser(): User? {
    var user: UserFromFirebase?
    try {
        val documentSnapshot : DocumentSnapshot! = db.collection(USERS).document(auth.uid!!).get().await()
        user = documentSnapshot.toObject()
        Log.d(FIRESTORE, msg: "DocumentSnapshot data: ${documentSnapshot.data}")
    } catch (e: Exception) {
        user = null
        Log.d(FIRESTORE, msg: "get failed with ", e)
    }
    return user?.toUser()
}
```

Figura 5-10 Proceso asíncorno ejecutado como código síncrono

Si no se utilizarán corrutinas el código quedaría de la siguiente forma:

```
suspend fun getUser(): User? {
    var user: UserFromFirebase?
    try {
        db.collection(USERS).document(auth.uid!!).get().addOnCompleteListener { task ->
            if (task.isSuccessful){
                Log.d(FIRESTORE, msg: "DocumentSnapshot data: ${task.result}")
                user = task.result?.toObject()
                return@addOnCompleteListener user?.toUser()
            }
            else
                Log.d(FIRESTORE, msg: "get failed with ", task.exception)
        }
    } catch (e: Exception) {
        user = null
        Log.d(FIRESTORE, msg: "get failed with ", e)
    }
}
```

Figura 5-11 Proceso asíncrono implementado con listeners

En este caso, el código incluso daría error y no se podría llamar a esta función esperando un retorno, ya que la función terminaría de ejecutarse y cuando Firestore proporcionara los datos el *listener* se dispararía, haciendo que la lectura de código no fuera secuencial. La acción que se quisiera realizar sobre los datos obtenidos tendría que hacerse dentro del

listener, haciendo más compleja la modularización de código. En situaciones en las que hay varios *listeners* anidados, las corrutinas facilitan mucho la lectura de código.

5.1.2 Patrones de diseño

Para implementar la aplicación se han seguido los siguientes patrones de diseño:

- **Observer:** a través de la librería LiveData, se generan observadores que se lanzan en la actividad/fragmento. Por otro lado, en el ViewModel se genera un objeto observable, de tal modo que cuando el ViewModel recibe los datos que ha solicitado al repositorio, los observadores son notificados de dicha actualización, toman los datos del ViewModel y actualizan la información que se muestra en pantalla.
- **Factory:** los ViewModel comparten muchas de sus propiedades y funciones, pero no todos comparten el mismo constructor. Por ello es necesario crear una factoría de ViewModel, que permite crearlos con diferentes argumentos para inyectar las dependencias o suministrar información extra.
- **Singleton:** las conexiones a la base de datos local, la base de datos remota y a la API son recursos que son costosos de generar y podrían afectar a la fluidez de la aplicación, por lo que es necesario que solo se genere una única instancia durante la ejecución de la aplicación. Por ello la creación del Repositorio sigue este patrón.
- **Service Locator:** gracias al patrón Singleton se consigue obtener un único Repositorio con el que acceder a los datos desde el ViewModel, pero para poder enviar una referencia a dicho repositorio, es necesario que existe un componente al que se pueda acceder desde cualquier punto de la aplicación.
- **Dependency Injection:** los ViewModel y el Repositorio tienen dependencias que pueden verse modificadas durante el desarrollo y mantenimiento del producto. Por ello estás deben ser inyectadas durante la creación del objeto. Gracias a este patrón también se permite crear dependencias que no utilicen servicios externos, para aislar componentes y poder realizar test unitarios o de integración sin tener en cuenta los posibles fallos de alguna de las dependencias.

5.2 Desarrollo de subsistemas

A continuación, se explica cómo se ha llevado a cabo el desarrollo de la aplicación para cada uno de los subsistemas, pasando primero por una breve descripción de elementos que todos ellos utilizan.

5.2.1 Elementos comunes a todos los subsistemas

La aplicación se ha desarrollado en el lenguaje de programación Kotlin, utilizando el entorno de desarrollo Android Studio. Este es utilizado para toda la lógica, mientras que para la definición de las vistas se utiliza XML.
Para la elaboración de todos los subsistemas se han utilizado varias librerías y componentes de Jetpack, además de otras desarrolladas por la propia Google y terceros, cuyos detalles se proporcionan en el Anexo.

5.2.2 Subsistema de Usuarios

El subsistema de usuarios se encarga de realizar la autenticación, creación, recuperación y almacenamiento de los datos de los usuarios.

Para la autenticación de los usuarios se utiliza Firebase Authentication. Este servicio de Firebase proporciona una librería para habilitar el inicio de sesión y registro de usuarios a través de muchos proveedores como pueden ser Facebook o Twitter, pero en este caso se utiliza la cuenta de Google habilitada en el dispositivo o una cuenta de email a elección del usuario. Además de ello, genera un identificador único para cada usuario que puede ser utilizado para relacionar sus datos con documentos en la base de datos remota. Esta funcionalidad es accedida por los ViewModel a través del repositorio.

Los datos del usuario son almacenados en dos localizaciones, una local y otra remota. Para el almacenamiento local se utiliza Room, de tal modo que la información del usuario una vez ha iniciado sesión queda almacenada en el dispositivo de forma permanente, a no ser que la aplicación sea desinstalada o se borren sus datos manualmente, de modo que cuando vaya a editar sus datos estos aparezcan con menor latencia. Por otro lado, la localización remota almacena el mismo contenido con respecto al usuario, pero permite que, si éste cambia de teléfono o elimina la aplicación por algún motivo, sus datos puedan recuperarse. Esta propiedad también habilita funcionalidad de autocompletado en la pantalla en la cual el usuario proporciona sus datos.

Toda esta funcionalidad del subsistema es llevada a cabo en el repositorio, pero es el ViewModel quien realmente solicita los datos. Por ello este subsistema utiliza un ViewModel, compartido por los cuatro fragmentos de este subsistema, que verifica el estado de autenticación del usuario en cuestión y solicita los datos del usuario en caso de que este se hubiera registrado con anterioridad.
Por último, la vista observa el ViewModel a la espera de los datos o del estado de autenticación y presenta los datos y los elementos para interactuar al usuario. Los cuatro fragmentos se han desarrollado para ello se muestran en las figuras 5-12 a 5-15.

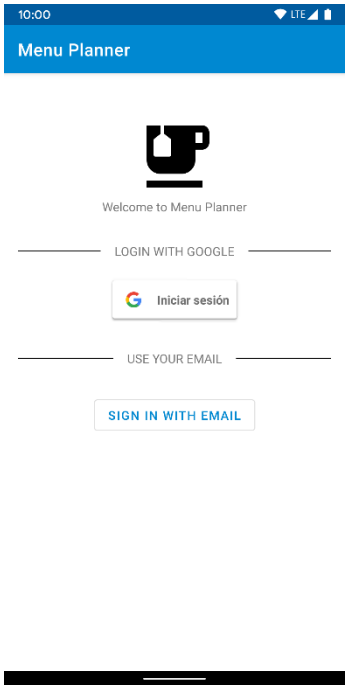


Figura 5-12 Fragmento de elección de inicio de sesión

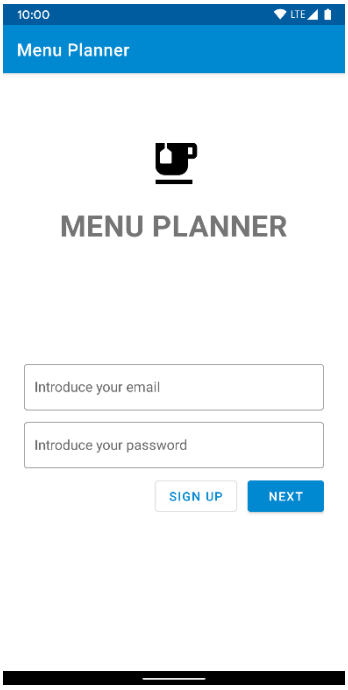


Figura 5-13 Fragmento de inicio de sesión con email

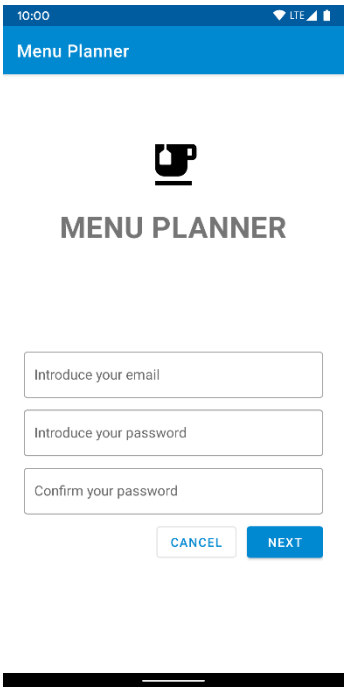


Figura 5-14 Fragmento de elección de registro por email

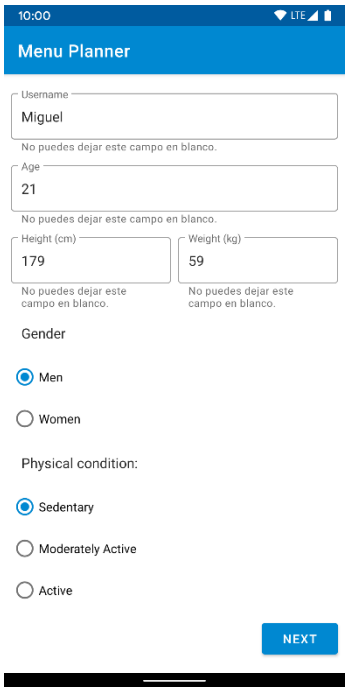


Figura 5-15 Fragmento de obtención de datos del usuario

En la Figura 5.15 se puede observar la funcionalidad de autocompletado mencionada anteriormente.
La navegación a través de estos fragmentos se ilustra en la figura 5-16

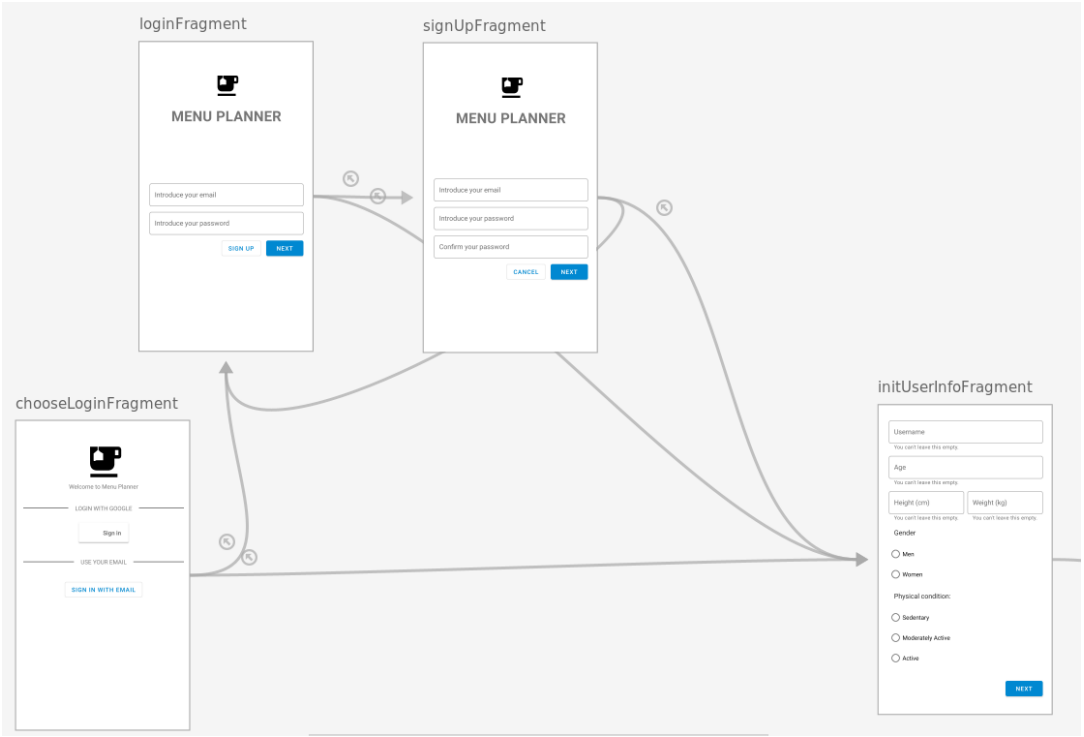


Figura 5-16 Navegación entre Fragmentos Subsistema Usuarios

5.2.2.1 Panel de control del usuario

Está sección permite al usuario modificar su información personal, consultar las recetas guardadas y acceder a la lista de la compra.

Esta sección de la aplicación utiliza un componente llamado ViewPager, el cual permite moverse entre fragmentos deslizando hacia los lados o utilizando los botones que aparecen en la zona superior, como en las tres figuras superiores. Desde las recetas favoritas se puede acceder a los detalles de la receta.
Todos estos fragmentos obtienen sus datos utilizando un ViewModel compartido que proporciona los datos a la vista siguiendo la arquitectura establecida. Para acceder a él se utiliza la barra de navegación inferior, utilizando el botón Profile, dando acceso a un nuevo fragmento. Los fragmentos que se muestran (ver figuras 5-17 a 5-20) son: fragmento con la lista de la compra, fragmento de datos del usuario, fragmento de recetas favoritas y fragmento de ajustes.

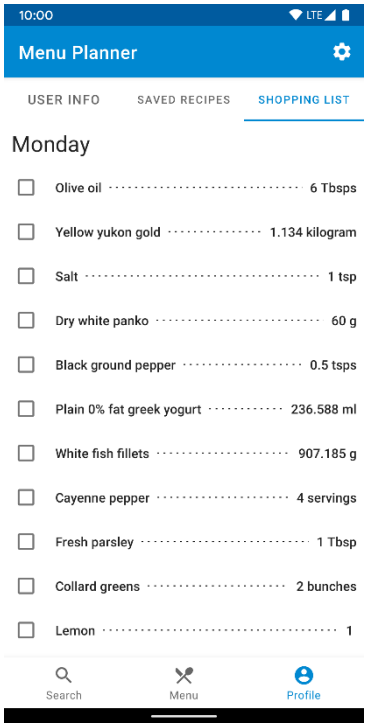


Figura 5-18 Fragmento con la lista de la compra de la semana

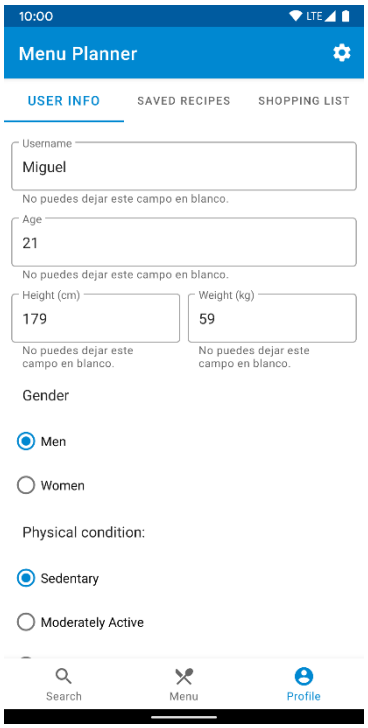


Figura 5-19 Fragmento con información del usuario

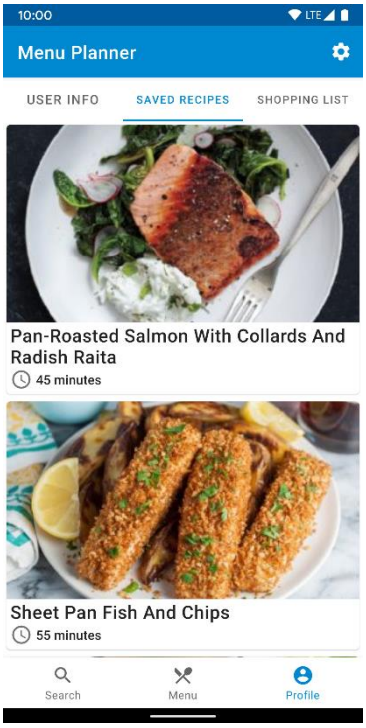


Figura 5-17 Fragmento con las recetas favoritas del usuario

Por último, a la pantalla de ajustes se accede haciendo una pulsación en la tuerca que aparece en la esquina superior derecha cuando el usuario se encuentra en el panel de control y permite o bien cerrar sesión o bien eliminar todos los datos del usuario en la nube y cerrar sesión.

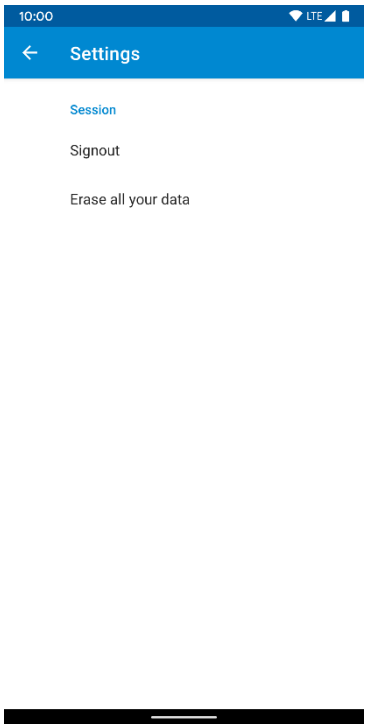


Figura 5-20 Pantalla de ajustes

5.2.3 Subsistema de Menú

El subsistema de menú se encarga de solicitar las recetas a la base de datos y mostrar los resultados. A continuación, se muestran los fragmentos del subsistema (figuras 5-21 a 5-26).

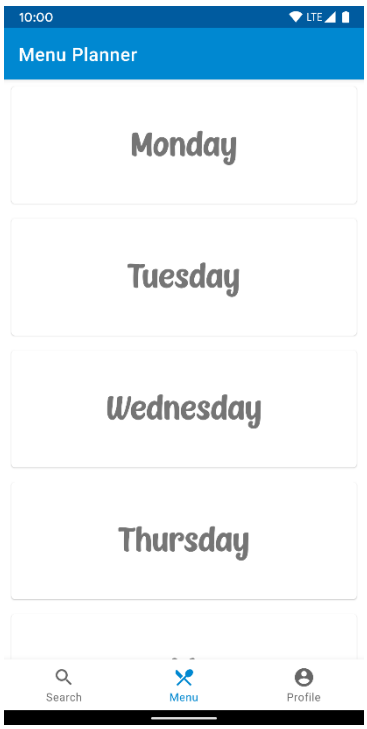


Figura 5-21 Fragmento días de la semana

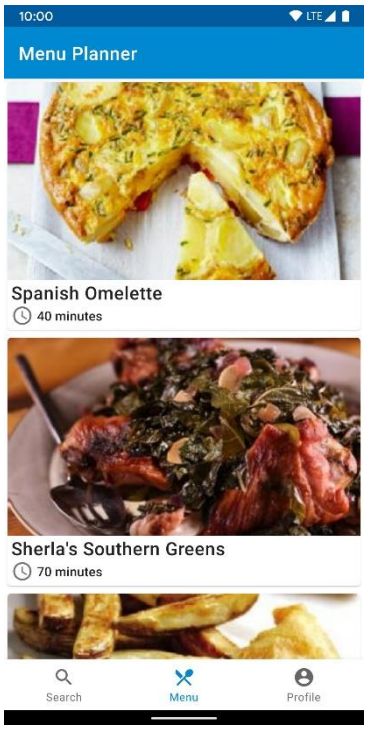


Figura 5-22 Fragmento recetas del día

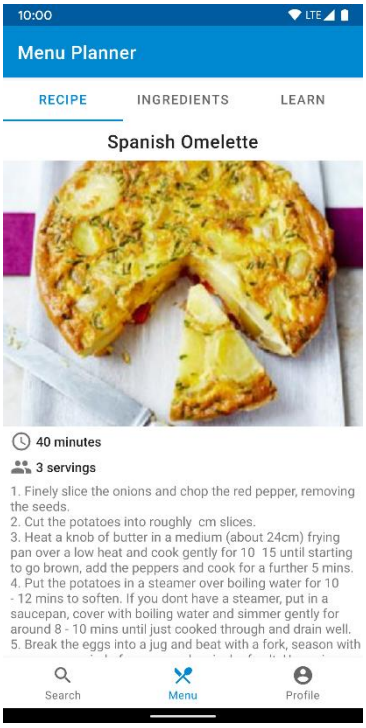


Figura 5-23 Fragmento receta detallada I

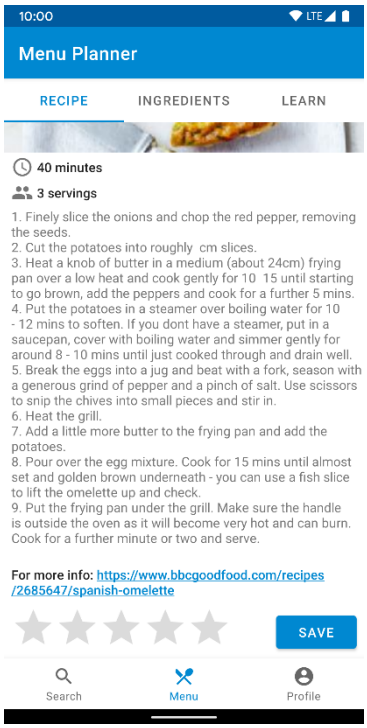


Figura 5-24 Fragmento receta detallada II

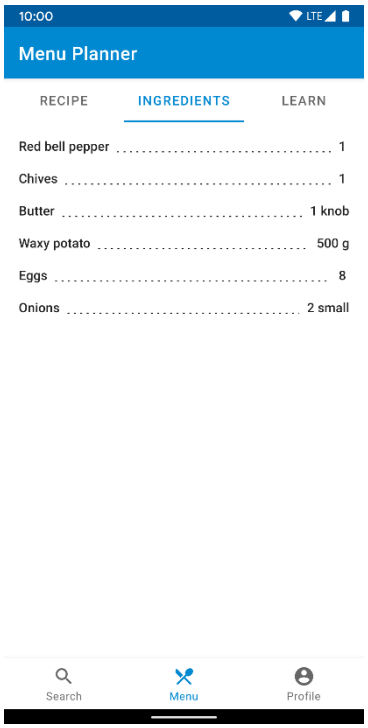


Figura 5-25 Fragmento ingredientes de receta

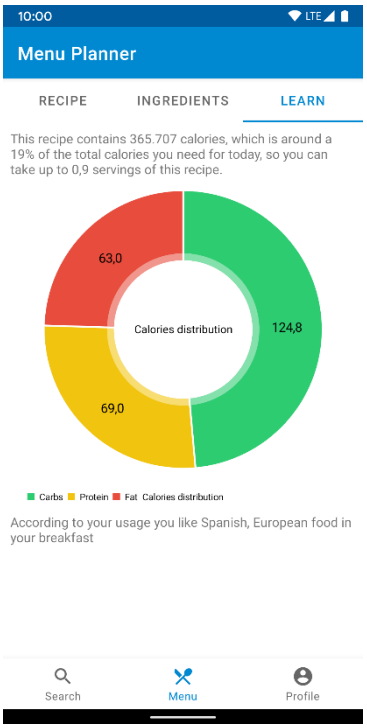


Figura 5-26 Fragmento con contenido explicativo

Utilizando el algoritmo de recomendación explicado anteriormente, se realiza el cálculo de calorías y los tipos de recetas que más le podrían gustar al usuario, se solicitan a Spoonacular las recetas en base a dichas condiciones y se almacenan en la base de datos.
Este proceso se dispara en dos escenarios distintos: el primero de ellos cuando el usuario inicia sesión y el segundo se lanza en segundo plano sin necesidad de abrir la aplicación, una vez han pasado siete días después de la última actualización.
Para implementar dicha funcionalidad se ha usado la librería Worker junto con las Corrutinas de Kotlin. De esta forma, si el teléfono está conectado a Internet en reposo, la batería no está por debajo del 15% y se ha cumplido un periodo de 7 días desde la última actualización, se carga en la aplicación un nuevo menú y se elimina el anterior.
La navegación entre estos fragmentos se muestra en la figura 5-27:

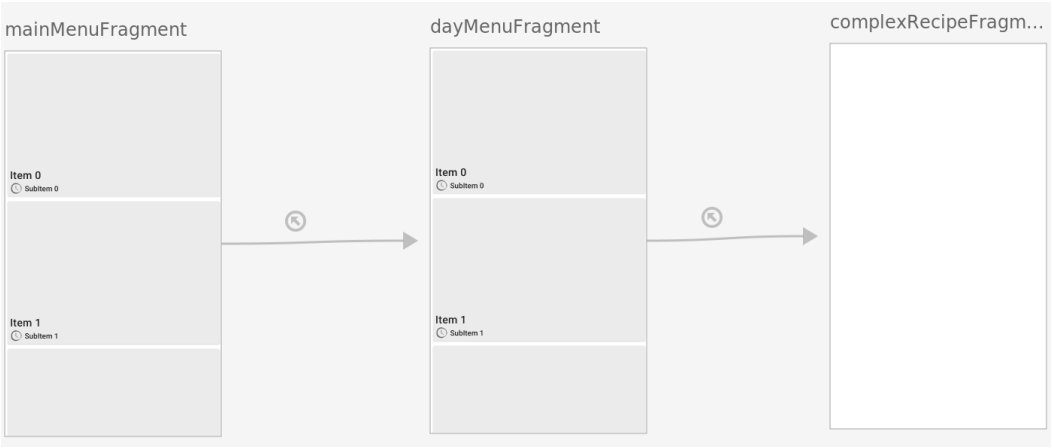


Figura 5-27 Navegación entre fragmentos subsistema Menú

La receta utiliza un ViewPager, que está contenido en lo que sería complexRecipeFragment, por lo que no aparece como un destino más en el diagrama de transiciones.
Como siempre, estos fragmentos actualizan sus datos con LiveData y ViewModel siguiendo la arquitectura establecida.
También es destacable el uso de corrutinas para optimizar tiempos de ejecución, aparte de las bondades mencionadas anteriormente. Al haber planteado las tres comidas como procesos independientes, se pueden lanzar tres hilos que se ejecutan en paralelo, donde cada uno solicita y convierte los datos de Spoonacular al modelo que se utiliza en la aplicación para la base de datos local.

```
private suspend fun refreshRecommendation() {
    withContext(Dispatchers.Default) { this: CoroutineScope
        launch { this: CoroutineScope
            val breakfast :String? = recommender.getBestFeature(rating?.breakfast, type: "breakfast")
            repository.refreshMenu(breakfast, category: "breakfast", recommender.bmr)
        }
        launch { this: CoroutineScope
            val lunch :String? = recommender.getBestFeature(rating?.lunch, type: "lunch")
            repository.refreshMenu(lunch, category: "lunch", recommender.bmr)
        }
        launch { this: CoroutineScope
            val dinner :String? = recommender.getBestFeature(rating?.dinner, type: "dinner")
            repository.refreshMenu(dinner, category: "dinner", recommender.bmr)
        } ^withContext
    }
}
```

Figura 5-28 Paralelización de tareas con corrutinas

Se utilizan corrutinas de tipo Default para obtener la característica más afín al usuario con respecto al tipo de comida y posteriormente de forma interna en la función refreshMenu se utilizan corrutinas IO.

5.2.4 Subsistema de Búsqueda

El subsistema de búsqueda se encarga de buscar recetas y mostrar los resultados a través de una lista, permitiendo también observar los detalles de éstas y guardarlas en la sección de recetas favoritas.
Está vista reutiliza la implementación de RecyclerView con DataBinding para mostrar las recetas, al igual que en el subsistema de menú. Para poder reutilizar esta vista se definen dos ficheros aparte, uno para definir la relación entre la vista recicladora y los elementos que la componen y otro para enlazar los datos a cada elemento de la vista. Para el primero de ellos se implementa ListAdapter con un comparador para que solo vuelva a dibujar aquellos elementos de la vista en los que se hayan modificados sus datos.
Por otro lado, para enlazar los datos se implementa una función por cada componente a mostrar, Por ejemplo, a la hora de cargar imágenes desde una URL, se tiene en cuenta que la obtención de la imagen pueda fallar y se suministra al componente una imagen de error en ese caso, como se muestra en la figura 5-29.

```
@BindingAdapter( ...value: "image")
fun ImageView.setImage(item: Recipe) {
    val imgUri :Uri! = item.image.toUri().buildUpon().scheme( scheme: "https").build()
    Glide.with(context).load(imgUri).apply(RequestOptions()
        .error(R.drawable.ic_outline_broken_image_24))
        .into( view: this)
}
```

Figura 5-29 Data Binding de imágenes con Glide

Este subsistema tiene una única vista que pasa por dos estados (ver figura 5-30) y se puede acceder a él utilizando la barra de navegación ubicada en la parte inferior.

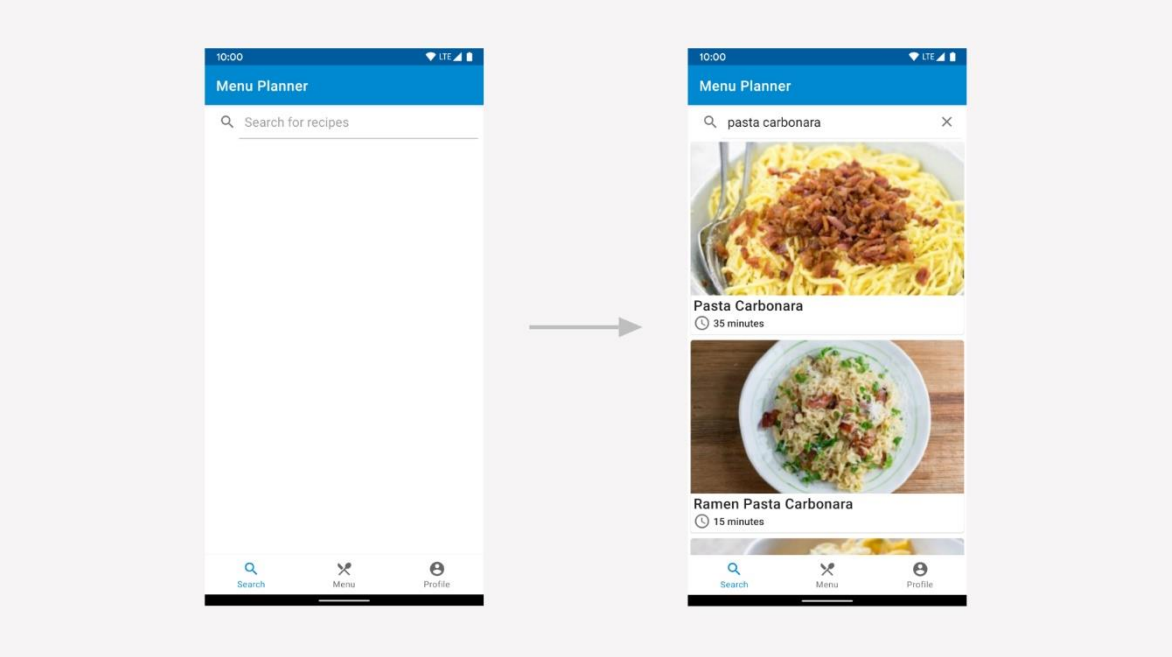


Figura 5-30 Estados fragmentos de búsqueda

Para realizar la búsqueda se utiliza el repositorio, que actúa de nexo con la API de Spoonacular, siguiendo el esquema habitual de Vista-Modelo-ViewModel.

5.2.5 Subsistema de Recomendación

El subsistema de recomendación se encarga de recoger los datos en cuanto a forma física del usuario y de procesar las puntuaciones para diferentes características de las recetas. De tal forma, se generan una serie de condiciones en base a estos dos procesos para realizar las consultas de recetas a la API de Spoonacular. Este subsistema es diferente con respecto a los anteriores, ya que no utiliza vistas, sino que se encarga de obtener los datos para proporcionárselos al modelo y de escuchar las solicitudes del subsistema de menú para renovar los platos tras el intervalo de 7 días.

6 Integración, pruebas y resultados

En el siguiente apartado se explica cómo se han llevado a cabo las pruebas y el resultado actual del producto desarrollado.

Dentro del desarrollo de aplicaciones Android existen distintas formas de testear la funcionalidad de una aplicación. Los proyectos desarrollados en Android Studio realizan una separación del código en tres secciones:

- main: contiene toda la lógica de negocio que permite el funcionamiento de la aplicación.
- test: contiene los tests que no necesitan del framework de Android para funcionar, permitiendo así la ejecución de tests con mayor velocidad, y son más fáciles de implementar. En esta sección es común realizar test unitarios, por ejemplo, sobre los métodos que proporcionan la funcionalidad del repositorio.
- androidTest: contiene los tests que necesitan del framework de Android y tienen en cuenta el contexto de la aplicación; por lo tanto, son más costosos de ejecutar y más complejos de desarrollar. Esta sección es comúnmente utilizada para test de casos de uso, como por ejemplo puede ser iniciar sesión en la aplicación.

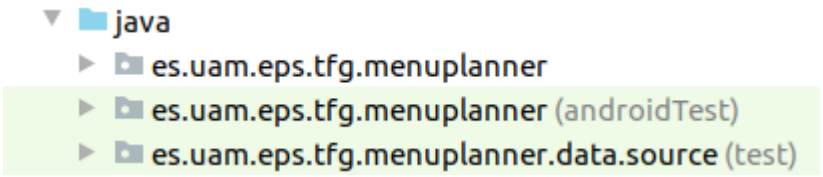


Figura 6-1 Organización de test en Android Studio

Para poder realizar pruebas de casos de uso que permiten simular las acciones del usuario se ha utilizado la librería Espresso, que simula acciones sobre la interfaz y comprueba que se carguen las vistas especificadas en la programación del test según va navegando por los fragmentos.

En concreto se ha realizado el test de todo el sistema de inicio de sesión hasta que se alcanza el menú principal, ya que este proceso utilizado los tres componentes principales de la arquitectura. Para ello se comprueban las tres líneas de ejecución posibles, inicio de sesión con Google, inicio de sesión con email y contraseña y, por último, registro con email y contraseña. Al ejecutar los tests se arranca el emulador y el sistema automáticamente rellena los formularios, comprueba que se lancen las vistas, se avance correctamente entre los fragmentos y finalmente deja la aplicación en el estado previo a la ejecución del test borrando los datos generados durante el mismo.

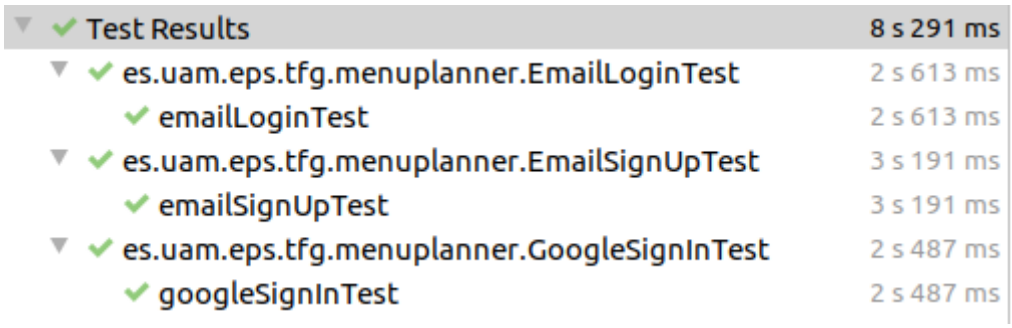


Figura 6-2 Resultados de test inicio de sesión

También se ha escrito un test para comprobar que el menú se vuelve a recargar correctamente, ya que la funcionalidad que realiza dicho proceso solo se ejecuta cada 7 días, lo cual hace que sea aún más importante que funcione sin errores. La aplicación se ha construido con el SDK 21, por lo que soporta teléfonos con Android 5.0.

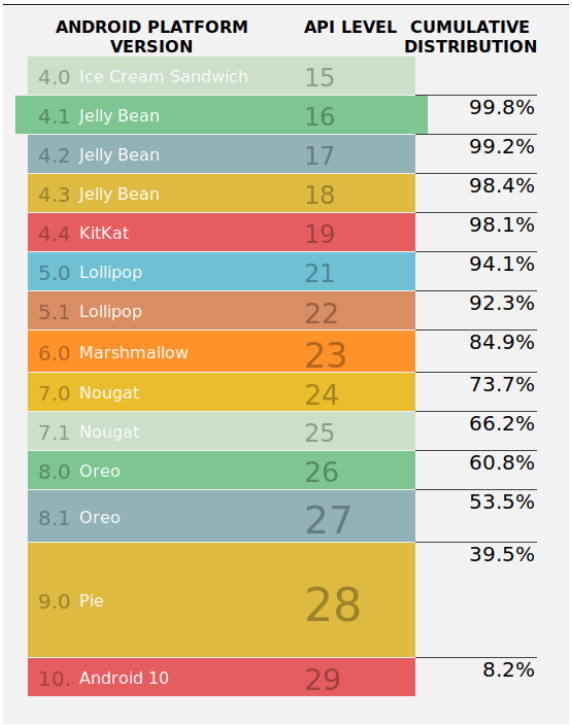


Figura 6-3 Distribución de las versiones de Android

Como se puede ver en la figura 6-3 generada por Android Studio, la aplicación podrá ser ejecutada en el 94,1% de los dispositivos Android en la actualidad.

En cuanto a consumo de recursos, Android Studio ofrece una herramienta llamada Profiler para comprobar el rendimiento.
En particular, la información sobre los recursos de la aplicación reflejada en la figura 6-4 se centra en su rendimiento general, desde el inicio de sesión, hasta la navegación por algunas de las pantallas.

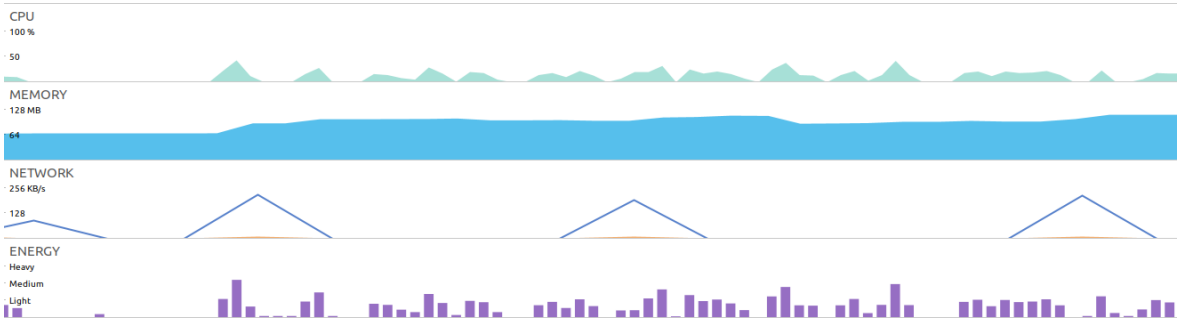


Figura 6-4 Monitor de recursos de la aplicacioón

Como se puede observar en la figura 6-4, la CPU solo alcanza picos de 50% de uso, manteniendóse bastante bajo el resto del tiempo y la RAM llega a un máximo de ocupación de 128MB. El consumo de red es muy bajo y en cuanto a energía se mueve entre niveles bajos y medios. El consumo no es alto y, teniendo en cuenta que en la actualidad la mayoría de dispositivos Android se mueve entre 3-4 Gb de RAM mínimo, ni siquiera el consumo máximo de memoria de la aplicación debería dar lugar a ralentizaciones.

7 Conclusiones y trabajo futuro

7.1 Conclusiones

Este trabajo tenía como objetivo crear una aplicación que ayude a planificar menús semanales recomendando platos adaptados a las necesidades de cada usuario, teniendo en cuenta sus características físicas y preferencias particulares, los nutrientes de los distintos alimentos y el momento del día en el que se recomienda consumirlos. El objetivo no era únicamente dar soporte a la recomendación personalizada de menús, sino también proporcionar funcionalidad complementaria para incorporar recetas, realizar una lista de la compra, automatizar todo el proceso e intentar que el usuario aprenda algunas cuestiones sobre alimentación y nutrición durante su uso.

Se ha diseñado una arquitectura robusta, con módulos bien diferenciados, lo cual facilita el mantenimiento de la aplicación y el testeo de los mismos. La utilización de fragmentos otorga flexibilidad a la interfaz y se ha orientado el diseño a evitar pulsaciones que puedan interactuar con los sistemas de navegación por gestos. El haber hecho un proyecto software centrado en el usuario ha aportado otro punto de vista al desarrollo de la aplicación. Se ha contado con los usuarios durante el análisis y diseño de la aplicación y, aunque se ha dejado para el futuro la implementación de algunas funcionalidades sugeridas por los usuarios, sí se han eliminado algunos componentes que generaban confusión.

El resultado final es una aplicación con una interfaz sencilla, que se ejecuta de forma fluida sin consumir muchos recursos y que cumple con la planificación y recomendación de menús a través de algoritmos que tienen en cuenta la salud física del usuario y sus gustos alimenticios.

Durante el desarrollo de este proyecto he podido trabajar desde la idea inicial, hasta las pruebas, poniendo en práctica los conocimientos adquiridos durante la carrera, a la vez que he podido utilizar también contenidos de programación aprendidos en (Codelabs-Android Kotlin Fundamentals, s.f.), (Codelabs-Advanced Android In Kotlin, s.f.) y (Material.io-Components, s.f.) consultando tutoriales en Internet. Además, he participado con el usuario

final durante la elaboración del trabajo, lo que me ha permitido ver el diseño de una aplicación desde un punto de vista diferente al del desarrollador.

7.2 Trabajo futuro

La implementación de fragmentos permite ampliar la aplicación a tamaños de pantalla más grandes, para extender su utilización a otros dispositivos con Android que no sean teléfonos, así como implementar personalización de temas en el diseño de la aplicación, como puede ser un modo oscuro. También sería interesante aplicar algunas de las ideas que han aportado los usuarios y que no han sido implementadas en la versión actual de la aplicación. Por otra parte, una vez que se haga un uso extendido de la misma, espero recibir retroalimentación sobre la satisfacción de los usuarios, para poder considerar sus valoraciones y sugerencias de mejora (tanto de funcionalidad como de interfaz) e incorporarlas en versiones sucesivas.

Spoonacular ofrece un servicio gratuito con un número muy limitado de solicitudes que, aun siendo suficientes para el desarrollo, no es un modelo viable para publicar la aplicación en la Play Store de Google a no ser que se utilice la API en su versión de pago. Por ello sería interesante eliminar esa dependencia y generar una base de datos poco a poco desde cero, alojada en Firestore, con recetas que los usuarios vayan subiendo, o incluso permitir el almacenamiento de menús enteros que estos generen. Al principio la aplicación no tendría mucha variedad de recetas, pero a largo plazo sería independiente de una API externa. Además, esto también abre la posibilidad a generar una red social donde puede haber usuarios con recetas mejor o peor valoradas, intercambiar recetas con amigos, posibilidad de que los usuarios puedan subir un vídeo con la preparación de las recetas, etc.
De todos modos, Firestore también es de pago, pero ofrece un plan gratuito con bastante margen que permite publicar la aplicación y ofrecer servicio mientras ésta no tenga un número de usuarios inicialmente muy grande. Por ello sería necesario implementar un sistema de remuneración a través de Google Ads o implementar una versión de pago con más funcionalidades para poder utilizar uno de los planes de pago de Firestore, hacer que la aplicación genere lo necesario para mejorar el plan de Firestore y así habilitar la acogida de un mayor número de usuarios.
En cuanto a seguridad, Room almacena los datos en plano en el dispositivo. Aunque por defecto solo la aplicación tiene permisos para acceder a dichos datos, si el teléfono tiene permisos de superusuario o acceso a depuración USB, un atacante podría leer la información del usuario, por lo que añadir encriptación a los datos sería una opción interesante, a costa de perder un poco de rendimiento.

También podría ser interesante desarrollar la aplicación para otras plataformas como iOS, ya sea desarrollando la aplicación con código nativo en Swift o utilizando algún sistema de programación híbrida como Xamarin o Flutter.

Referencias

Android Developers. (s.f.). Obtenido de https://developer.android.com/guide/components/activities/activity-lifecycle

Android Developers. (s.f.). Obtenido de https://developer.android.com/guide/components/activities/activity-lifecycle

Android Developers. (s.f.). Obtenido de https://developer.android.com/topic/libraries/architecture/viewmodel

Codelabs-Advanced Android In Kotlin. (s.f.). Obtenido de https://codelabs.developers.google.com/advanced-android-kotlin-training/

Codelabs-Android Kotlin Fundamentals. (s.f.). Obtenido de https://codelabs.developers.google.com/android-kotlin-fundamentals/

David Elsweiler, C. T. (2017). Exploiting Food Choice Biases for Healthier Recipe Recommendation. *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '17)* (págs. 575–584). New York, NY, USA: Association for Computing Machinery.

David Elsweiler, M. H. (Septiembre de 2015). *acm.org*. Obtenido de https://dl.acm.org/doi/abs/10.1145/2792838.2799665

Globalnews. (s.f.). Obtenido de https://globalnews.ca/news/3615212/this-is-what-your-breakfast-lunch-and-dinner-calories-actually-look-like/#:~:text=Although%20every%20person's%20daily%20caloric,each%20for%20lunch%20and%20dinner.

Harris-Benedict. (s.f.). *Microhealthllc*. Obtenido de https://www.microhealthllc.com/how-to-calculate-basal-metabolic-rate-bmr/

M. Chen, X. J. (2019). Eating healthier: Exploring nutrition information for healthier recipe recommendation. *Information Processing & Management*.

Material.io-Components. (s.f.). Obtenido de https://material.io/components

Shichao Zang, X. L. (12 de Abril de 2017). *IEEE*. Obtenido de https://ieeexplore.ieee.org/document/7898482?denied=

Taejin Jung, J. H. (2019). *Influence of school-based nutrition education program on healthy eating literacy and healthy food choice among primary school children.* International Journal of Health Promotion and Education.

Teh Lee Cheng, U. K. (18 de Diciembre de 2014). *IEEExplore.* Obtenido de https://ieeexplore.ieee.org/document/6986011?denied=

Uccs. (s.f.). Obtenido de https://www.uccs.edu/healthcircle/sites/healthcircle/files/inline-files/Meal%20Planning.pdf

Vivek M.B., M. N. (2018). Machine Learning Based Food Recipe Recommendation System. *Lecture Notes in Networks and Systems, vol 14.* Springer, Singapore.

Glosario

API	Application Programming Interface
Android	Sistema operativo para dispositivos móviles
Android Jetpack	Conjunto de librerías para desarrollo en Android
Android Studio	Entorno de desarrollo para aplicaciones Android
Firebase	Plataforma para desarrollo web y móvil desarrollada por Google
Firestore	Base de datos NoSQL alojada en Firebase
Flutter	Entorno de desarrollo para programación híbrida de aplicaciones Android, iOS y Web a través de Dart.
Listener	Capturador de eventos generados por una tarea asíncrona
Xamarin	Entorno de desarrollo para programación híbrido de aplicaciones Android e iOS a través de .NET y C#

Anexo

Librerías destacadas

LiveData: librería encargada de implementar el patrón Observer de forma que este sea consciente del ciclo de vida de elementos de una aplicación Android como pueden ser los fragmentos. Esto elimina perdidas de memoria, ya que los objetos se liberan junto con el ciclo de vida. En esta aplicación se han utilizado para las comunicaciones entre el ViewModel y las vistas.

Navigation: librería encargada de gestionar la pila de fragmentos y las acciones de navegación entre ellos. En esta aplicación se encarga de realizar todas las transiciones entre fragmentos, definiendo las acciones entre fragmentos, gestionando información que se envía entre ellos y controlando la pila de fragmentos para la acción de ir atrás.

DataBinding y Propiedades Sintéticas de Kotlin: las vistas dibujadas por las actividades/fragmentos se representan en XML y se acceden desde el fragmento utilizando DataBinding o propiedades sintéticas. De esta forma se proporciona contenido y la vista y se configuran aspectos relacionados con el comportamiento de dichas vistas y su estética.

RecyclerView: las vistas que presentan una lista de elementos se implementan utilizando este componente que se encarga de reutilizar las vistas generadas, siguiendo el patrón Adapter, modificando la información que contienen para así optimizar el uso de memoria y los tiempos de respuesta.